

---

# **ADB Shell Documentation**

*Release 0.3.0*

**Jeff Irion**

**Dec 22, 2020**



# CONTENTS

<b>1 adb_shell</b>	<b>1</b>
1.1 adb_shell package . . . . .	1
<b>2 Installation</b>	<b>49</b>
2.1 Async . . . . .	49
2.2 USB Support (Experimental) . . . . .	49
<b>3 Example Usage</b>	<b>51</b>
<b>4 Indices and tables</b>	<b>53</b>
<b>Python Module Index</b>	<b>55</b>
<b>Index</b>	<b>57</b>



## ADB\_SHELL

### 1.1 adb\_shell package

#### 1.1.1 Subpackages

##### adb\_shell.auth package

##### Submodules

##### adb\_shell.auth.keygen module

This file implements encoding and decoding logic for Android's custom RSA public key binary format. Public keys are stored as a sequence of little-endian 32 bit words. Note that Android only supports little-endian processors, so we don't do any byte order conversions when parsing the binary struct.

Structure from: [https://github.com/aosp-mirror/platform\\_system\\_core/blob/c55fab4a59cfa461857c6a61d8a0f1ae4591900c/libcrypto\\_utils/android\\_pubkey.c](https://github.com/aosp-mirror/platform_system_core/blob/c55fab4a59cfa461857c6a61d8a0f1ae4591900c/libcrypto_utils/android_pubkey.c)

```
typedef struct RSAPublicKey {
    // Modulus length. This must be ANDROID_PUBKEY_MODULUS_SIZE_WORDS
    uint32_t modulus_size_words;

    // Precomputed montgomery parameter: -1 / n[0] mod 2^32
    uint32_t n0inv;

    // RSA modulus as a little-endian array
    uint8_t modulus[ANDROID_PUBKEY_MODULUS_SIZE];

    // Montgomery parameter R^2 as a little-endian array of little-endian words
    uint8_t rr[ANDROID_PUBKEY_MODULUS_SIZE];

    // RSA modulus: 3 or 65537
    uint32_t exponent;
} RSAPublicKey;
```

#### Contents

- `_to_bytes()`
- `decode_pubkey()`
- `decode_pubkey_file()`

- `encode_pubkey()`
- `get_user_info()`
- `keygen()`
- `write_public_keyfile()`

`adb_shell.auth.keygen.ANDROID_PUBKEY_MODULUS_SIZE = 256`  
Size of an RSA modulus such as an encrypted block or a signature.

`adb_shell.auth.keygen.ANDROID_PUBKEY_MODULUS_SIZE_WORDS = 64`  
Size of the RSA modulus in words.

`adb_shell.auth.keygen.ANDROID_RSAPUBLICKEY_STRUCT = '<LL256s256sL'`  
Python representation of “struct RSAPublicKey”

`adb_shell.auth.keygen._to_bytes(n, length, endianness='big')`  
Partial python2 compatibility with `int.to_bytes`

<https://stackoverflow.com/a/20793663>

**Parameters**

- `n (TODO)` – TODO
- `length (TODO)` – TODO
- `endianness (str, TODO)` – TODO

**Returns** TODO

**Return type** TODO

`adb_shell.auth.keygen.decode_pubkey(public_key)`  
Decode a public RSA key stored in Android’s custom binary format.

**Parameters** `public_key (TODO)` – TODO

`adb_shell.auth.keygen.decode_pubkey_file(public_key_path)`  
TODO

**Parameters** `public_key_path (str)` – TODO

`adb_shell.auth.keygen.encode_pubkey(private_key_path)`  
Encodes a public RSA key into Android’s custom binary format.

**Parameters** `private_key_path (str)` – TODO

**Returns** TODO

**Return type** TODO

`adb_shell.auth.keygen.get_user_info()`  
TODO

**Returns** ' <username>@<hostname>

**Return type** str

`adb_shell.auth.keygen.keygen(filepath)`  
Generate an ADB public/private key pair.

- The private key is stored in `filepath`.
- The public key is stored in `filepath + '.pub'`

(Existing files will be overwritten.)

**Parameters** `filepath` (*str*) – File path to write the private/public keypair

`adb_shell.auth.keygen.write_public_keyfile` (*private\_key\_path, public\_key\_path*)

Write a public keyfile to `public_key_path` in Android's custom RSA public key format given a path to a private keyfile.

**Parameters**

- `private_key_path` (*TODO*) – TODO
- `public_key_path` (*TODO*) – TODO

## adb\_shell.auth.sign\_cryptography module

ADB authentication using the `cryptography` package.

### Contents

- `CryptographySigner`
  - `CryptographySigner.GetPublicKey()`
  - `CryptographySigner.Sign()`

**class** `adb_shell.auth.sign_cryptography.CryptographySigner` (*rsa\_key\_path*)

Bases: `object`

AuthSigner using `cryptography.io`.

**Parameters** `rsa_key_path` (*str*) – The path to the private key.

**public\_key**

The contents of the public key file

**Type** `str`

**rsa\_key**

The loaded private key

**Type** `cryptography.hazmat.backends.openssl.rsa._RSAPrivateKey`

**GetPublicKey** ()

Returns the public key in PEM format without headers or newlines.

**Returns** `self.public_key` – The contents of the public key file

**Return type** `str`

**Sign** (*data*)

Signs given data using a private key.

**Parameters** `data` (*TODO*) – TODO

**Returns** The signed data

**Return type** `TODO`

## adb\_shell.auth.sign\_pycryptodome module

ADB authentication using `pycryptodome`.

## Contents

- *PycryptodomeAuthSigner*
  - *PycryptodomeAuthSigner.GetPublicKey()*
  - *PycryptodomeAuthSigner.Sign()*

**class** adb\_shell.auth.sign\_pycryptodome.**PycryptodomeAuthSigner** (*rsa\_key\_path=None*)  
Bases: object

AuthSigner using the pycryptodome package.

**Parameters** *rsa\_key\_path* (*str, None*) – The path to the private key

**public\_key**

The contents of the public key file

**Type** str

**rsa\_key**

The contents of the private key

**Type** Crypto.PublicKey.RSA.RsaKey

**GetPublicKey()**

Returns the public key in PEM format without headers or newlines.

**Returns** *self.public\_key* – The contents of the public key file

**Return type** str

**Sign** (*data*)

Signs given data using a private key.

**Parameters** *data* (*bytes, bytearray*) – The data to be signed

**Returns** The signed data

**Return type** bytes

## adb\_shell.auth.sign\_pythonrsa module

ADB authentication using the *rsa* package.

## Contents

- *\_Accum*
  - *\_Accum.digest()*
  - *\_Accum.update()*
- *\_load\_rsa\_private\_key()*
- *PythonRSASigner*
  - *PythonRSASigner.FromRSAKeyPath()*
  - *PythonRSASigner.GetPublicKey()*
  - *PythonRSASigner.Sign()*



**class** adb\_shell.auth.sign\_pythonrsa.**PythonRSASigner** (*pub=None, priv=None*)

Bases: object

Implements adb\_protocol.AuthSigner using <http://stuvel.eu/rsa>.

#### Parameters

- **pub** (*str, None*) – The contents of the public key file
- **priv** (*str, None*) – The path to the private key

#### priv\_key

The loaded private key

**Type** rsa.key.PrivateKey

#### pub\_key

The contents of the public key file

**Type** str, None

**classmethod** **FromRSAKeyPath** (*rsa\_key\_path*)

Create a *PythonRSASigner* instance using the provided private key.

**Parameters** **rsa\_key\_path** (*str*) – The path to the private key; the public key must be `rsa_key_path + '.pub'`.

**Returns** A *PythonRSASigner* with private key `rsa_key_path` and public key `rsa_key_path + '.pub'`

**Return type** *PythonRSASigner*

**GetPublicKey** ()

Returns the public key in PEM format without headers or newlines.

**Returns** **self.pub\_key** – The contents of the public key file, or `None` if a public key was not provided.

**Return type** str, None

**Sign** (*data*)

Signs given data using a private key.

**Parameters** **data** (*bytes*) – The data to be signed

**Returns** The signed data

**Return type** bytes

**class** adb\_shell.auth.sign\_pythonrsa.**\_Accum**

Bases: object

A fake hashing algorithm.

The Python `rsa` lib hashes all messages it signs. ADB does it already, we just need to slap a signature on top of already hashed message. Introduce a “fake” hashing algo for this.

#### **\_buf**

A buffer for storing data before it is signed

**Type** bytes

**digest** ()

Return the digest value as a string of binary data.

**Returns** **self.\_buf** – `self._buf`

**Return type** bytes

**update** (*msg*)

Update this hash object's state with the provided *msg*.

**Parameters** *msg* (*bytes*) – The message to be appended to `self._buf`

`adb_shell.auth.sign_pythonrsa._load_rsa_private_key` (*pem*)

PEM encoded PKCS#8 private key -> `rsa.PrivateKey`.

ADB uses private RSA keys in pkcs#8 format. The `rsa` library doesn't support them natively. Do some ASN unwrapping to extract naked RSA key (in der-encoded form).

See:

- <https://www.ietf.org/rfc/rfc2313.txt>
- <http://superuser.com/a/606266>

**Parameters** *pem* (*str*) – The private key to be loaded

**Returns** The loaded private key

**Return type** `rsa.key.PrivateKey`

## Module contents

### `adb_shell.transport` package

#### Submodules

#### `adb_shell.transport.base_transport` module

A base class for transports used to communicate with a device.

- `BaseTransport`
  - `BaseTransport.bulk_read()`
  - `BaseTransport.bulk_write()`
  - `BaseTransport.close()`
  - `BaseTransport.connect()`

**class** `adb_shell.transport.base_transport.BaseTransport`

Bases: `abc.ABC`

A base transport class.

`_abc_impl` = `<_abc_data object>`

**abstract** `bulk_read` (*numbytes*, *transport\_timeout\_s*)

Read data from the device.

**Parameters**

- **numbytes** (*int*) – The maximum amount of data to be received
- **transport\_timeout\_s** (*float*, *None*) – A timeout for the read operation

**Returns** The received data

**Return type** bytes

**abstract bulk\_write** (*data, transport\_timeout\_s*)

Send data to the device.

**Parameters**

- **data** (*bytes*) – The data to be sent
- **transport\_timeout\_s** (*float, None*) – A timeout for the write operation

**Returns** The number of bytes sent

**Return type** int

**abstract close** ()

Close the connection.

**abstract connect** (*transport\_timeout\_s*)

Create a connection to the device.

**Parameters** **transport\_timeout\_s** (*float, None*) – A connection timeout

## adb\_shell.transport.base\_transport\_async module

A base class for transports used to communicate with a device.

- *BaseTransportAsync*
  - *BaseTransportAsync.bulk\_read()*
  - *BaseTransportAsync.bulk\_write()*
  - *BaseTransportAsync.close()*
  - *BaseTransportAsync.connect()*

**class** adb\_shell.transport.base\_transport\_async.**BaseTransportAsync**

Bases: abc.ABC

A base transport class.

**\_abc\_impl** = <\_abc\_data object>

**abstract async bulk\_read** (*numbytes, transport\_timeout\_s*)

Read data from the device.

**Parameters**

- **numbytes** (*int*) – The maximum amount of data to be received
- **transport\_timeout\_s** (*float, None*) – A timeout for the read operation

**Returns** The received data

**Return type** bytes

**abstract async bulk\_write** (*data, transport\_timeout\_s*)

Send data to the device.

**Parameters**

- **data** (*bytes*) – The data to be sent
- **transport\_timeout\_s** (*float, None*) – A timeout for the write operation

**Returns** The number of bytes sent

**Return type** int

**abstract async close()**

Close the connection.

**abstract async connect(*transport\_timeout\_s*)**

Create a connection to the device.

**Parameters** *transport\_timeout\_s* (*float, None*) – A connection timeout

## adb\_shell.transport.tcp\_transport module

A class for creating a socket connection with the device and sending and receiving data.

- *TcpTransport*

- *TcpTransport.bulk\_read()*

- *TcpTransport.bulk\_write()*

- *TcpTransport.close()*

- *TcpTransport.connect()*

**class** adb\_shell.transport.tcp\_transport.**TcpTransport** (*host, port=5555*)

Bases: *adb\_shell.transport.base\_transport.BaseTransport*

TCP connection object.

**Parameters**

- **host** (*str*) – The address of the device; may be an IP address or a host name

- **port** (*int*) – The device port to which we are connecting (default is 5555)

**connection**

A socket connection to the device

**Type** socket.socket, None

**host**

The address of the device; may be an IP address or a host name

**Type** str

**port**

The device port to which we are connecting (default is 5555)

**Type** int

**\_abc\_impl** = **<\_abc\_data object>**

**bulk\_read** (*numbytes, transport\_timeout\_s*)

Receive data from the socket.

**Parameters**

- **numbytes** (*int*) – The maximum amount of data to be received

- **transport\_timeout\_s** (*float, None*) – When the timeout argument is omitted, `select.select` blocks until at least one file descriptor is ready. A time-out value of zero specifies a poll and never blocks.

**Returns** The received data

**Return type** bytes

**Raises** *TcpTimeoutException* – Reading timed out.

**bulk\_write** (*data*, *transport\_timeout\_s*)

Send data to the socket.

**Parameters**

- **data** (*bytes*) – The data to be sent
- **transport\_timeout\_s** (*float*, *None*) – When the timeout argument is omitted, `select.select` blocks until at least one file descriptor is ready. A time-out value of zero specifies a poll and never blocks.

**Returns** The number of bytes sent

**Return type** `int`

**Raises** `TcpTimeoutException` – Sending data timed out. No data was sent.

**close** ()

Close the socket connection.

**connect** (*transport\_timeout\_s*)

Create a socket connection to the device.

**Parameters** **transport\_timeout\_s** (*float*, *None*) – Set the timeout on the socket instance

## adb\_shell.transport.tcp\_transport\_async module

A class for creating a socket connection with the device and sending and receiving data.

- `TcpTransportAsync`
  - `TcpTransportAsync.bulk_read()`
  - `TcpTransportAsync.bulk_write()`
  - `TcpTransportAsync.close()`
  - `TcpTransportAsync.connect()`

**class** `adb_shell.transport.tcp_transport_async.TcpTransportAsync` (*host*, *port=5555*)

Bases: `adb_shell.transport.base_transport_async.BaseTransportAsync`

TCP connection object.

**Parameters**

- **host** (*str*) – The address of the device; may be an IP address or a host name
- **port** (*int*) – The device port to which we are connecting (default is 5555)

**\_host**

The address of the device; may be an IP address or a host name

**Type** `str`

**\_port**

The device port to which we are connecting (default is 5555)

**Type** `int`

**\_reader**

Object for reading data from the socket

**Type** `StreamReader`, `None`

**\_writer**

Object for writing data to the socket

**Type** StreamWriter, None

**\_abc\_impl** = <\_abc\_data object>

**async bulk\_read** (*numbytes, transport\_timeout\_s*)

Receive data from the socket.

**Parameters**

- **numbytes** (*int*) – The maximum amount of data to be received
- **transport\_timeout\_s** (*float, None*) – Timeout for reading data from the socket; if it is *None*, then it will block until the read operation completes

**Returns** The received data

**Return type** bytes

**Raises** *TcpTimeoutException* – Reading timed out.

**async bulk\_write** (*data, transport\_timeout\_s*)

Send data to the socket.

**Parameters**

- **data** (*bytes*) – The data to be sent
- **transport\_timeout\_s** (*float, None*) – Timeout for writing data to the socket; if it is *None*, then it will block until the write operation completes

**Returns** The number of bytes sent

**Return type** int

**Raises** *TcpTimeoutException* – Sending data timed out. No data was sent.

**async close** ()

Close the socket connection.

**async connect** (*transport\_timeout\_s*)

Create a socket connection to the device.

**Parameters** **transport\_timeout\_s** (*float, None*) – Timeout for connecting to the socket; if it is *None*, then it will block until the operation completes

**adb\_shell.transport.usb\_transport module**

A class for creating a USB connection with the device and sending and receiving data.

**Warning:** USB support is an experimental feature.

- *get\_interface()*
- *interface\_matcher()*
- *UsbTransport*
  - *UsbTransport.\_find()*
  - *UsbTransport.\_find\_and\_open()*

```

- UsbTransport._find_devices()
- UsbTransport._find_first()
- UsbTransport._flush_buffers()
- UsbTransport._open()
- UsbTransport._port_path_matcher()
- UsbTransport._serial_matcher()
- UsbTransport._timeout()
- UsbTransport.bulk_read()
- UsbTransport.bulk_write()
- UsbTransport.close()
- UsbTransport.connect()
- UsbTransport.port_path
- UsbTransport.serial_number
- UsbTransport.usb_info

```

```
adb_shell.transport.usb_transport.DEFAULT_TIMEOUT_S = 10
Default timeout
```

```
class adb_shell.transport.usb_transport.UsbTransport (device, setting,
                                                    usb_info=None, de-
                                                    fault_transport_timeout_s=None)
```

Bases: *adb\_shell.transport.base\_transport.BaseTransport*

USB communication object. Not thread-safe.

Handles reading and writing over USB with the proper endpoints, exceptions, and interface claiming.

#### Parameters

- **device** (*usb1.USBDevice*) – libusb\_device to connect to.
- **setting** (*usb1.USBInterfaceSetting*) – libusb setting with the correct endpoints to communicate with.
- **usb\_info** (*TODO, None*) – String describing the usb path/serial/device, for debugging.
- **default\_transport\_timeout\_s** (*TODO, None*) – Timeout in seconds for all I/O.

**`_default_transport_timeout_s`**  
Timeout in seconds for all I/O.

**Type** *TODO, None*

**`_device`**  
libusb\_device to connect to.

**Type** *TODO*

**`_transport`**  
*TODO*

**Type** *TODO*

**`_interface_number`**  
*TODO*

Type TODO

`_max_read_packet_len`  
TODO

Type TODO

`_read_endpoint`  
TODO

Type TODO

`_setting`  
libusb setting with the correct endpoints to communicate with.

Type TODO

`_usb_info`  
String describing the usb path/serial/device, for debugging.

Type TODO

`_write_endpoint`  
TODO

Type TODO, None

`_HANDLE_CACHE` = <WeakValueDictionary>

`_HANDLE_CACHE_LOCK` = <unlocked `_thread.lock` object>

`_abc_impl` = <`_abc_data` object>

`classmethod _find` (*setting\_matcher*, *port\_path=None*, *serial=None*, *de-*  
*fault\_transport\_timeout\_s=None*)

Gets the first device that matches according to the keyword args.

Parameters

- `setting_matcher` (*TODO*) – TODO
- `port_path` (*TODO*, *None*) – TODO
- `serial` (*TODO*, *None*) – TODO
- `default_transport_timeout_s` (*TODO*, *None*) – TODO

Returns TODO

Return type TODO

`classmethod _find_and_open` (*setting\_matcher*, *port\_path=None*, *serial=None*, *de-*  
*fault\_transport\_timeout\_s=None*)

TODO

Parameters

- `setting_matcher` (*TODO*) – TODO
- `port_path` (*TODO*, *None*) – TODO
- `serial` (*TODO*, *None*) – TODO
- `default_transport_timeout_s` (*TODO*, *None*) – TODO

Returns `dev` – TODO

Return type TODO



**classmethod** `_find_devices` (*setting\_matcher*, *device\_matcher=None*, *usb\_info=""*, *default\_transport\_timeout\_s=None*)  
 \_find and yield the devices that match.

**Parameters**

- **setting\_matcher** (*TODO*) – Function that returns the setting to use given a `usb1.USBDevice`, or `None` if the device doesn't have a valid setting.
- **device\_matcher** (*TODO*, *None*) – Function that returns `True` if the given `UsbTransport` is valid. `None` to match any device.
- **usb\_info** (*str*) – Info string describing device(s).
- **default\_transport\_timeout\_s** (*TODO*, *None*) – Default timeout of commands in seconds.

**Yields** *TODO* – `UsbTransport` instances

**classmethod** `_find_first` (*setting\_matcher*, *device\_matcher=None*, *usb\_info=""*, *default\_transport\_timeout\_s=None*)  
 Find and return the first matching device.

**Parameters**

- **setting\_matcher** (*TODO*) – Function that returns the setting to use given a `usb1.USBDevice`, or `None` if the device doesn't have a valid setting.
- **device\_matcher** (*TODO*) – Function that returns `True` if the given `UsbTransport` is valid. `None` to match any device.
- **usb\_info** (*str*) – Info string describing device(s).
- **default\_transport\_timeout\_s** (*TODO*, *None*) – Default timeout of commands in seconds.

**Returns** An instance of `UsbTransport`

**Return type** `TODO`

**Raises** `adb_shell.exceptions.DeviceNotFoundError` – Raised if the device is not available.

**flush\_buffers** ()  
`TODO`

**Raises** `adb_shell.exceptions.UsbReadFailedError` – `TODO`

**open** ()  
 Opens the USB device for this setting, and claims the interface.

**classmethod** `_port_path_matcher` (*port\_path*)  
 Returns a device matcher for the given port path.

**Parameters** `port_path` (*TODO*) – `TODO`

**Returns** `TODO`

**Return type** `function`

**classmethod** `_serial_matcher` (*serial*)  
 Returns a device matcher for the given serial.

**Parameters** `serial` (*TODO*) – `TODO`

**Returns** `TODO`

**Return type** function

**`_timeout_ms`** (*transport\_timeout\_s*)  
TODO

**Returns** TODO

**Return type** TODO

**`bulk_read`** (*numbytes, transport\_timeout\_s=None*)  
Receive data from the USB device.

**Parameters**

- **`numbytes`** (*int*) – The maximum amount of data to be received
- **`transport_timeout_s`** (*float, None*) – When the timeout argument is omitted, `select.select` blocks until at least one file descriptor is ready. A time-out value of zero specifies a poll and never blocks.

**Returns** The received data

**Return type** bytes

**Raises** `adb_shell.exceptions.UsbReadFailedError` – Could not receive data

**`bulk_write`** (*data, transport\_timeout\_s=None*)  
Send data to the USB device.

**Parameters**

- **`data`** (*bytes*) – The data to be sent
- **`transport_timeout_s`** (*float, None*) – When the timeout argument is omitted, `select.select` blocks until at least one file descriptor is ready. A time-out value of zero specifies a poll and never blocks.

**Returns** The number of bytes sent

**Return type** int

**Raises**

- `adb_shell.exceptions.UsbWriteFailedError` – This transport has been closed, probably due to another being opened
- `adb_shell.exceptions.UsbWriteFailedError` – Could not send data

**`close`** ()  
Close the USB connection.

**`connect`** (*transport\_timeout\_s=None*)  
Create a USB connection to the device.

**Parameters** **`transport_timeout_s`** (*float, None*) – Set the timeout on the USB instance

**classmethod** **`find_adb`** (*serial=None, port\_path=None, default\_transport\_timeout\_s=None*)  
TODO

**Parameters**

- **`serial`** (*TODO*) – TODO
- **`port_path`** (*TODO*) – TODO

- **default\_transport\_timeout\_s** (*TODO*, *None*) – Default timeout of commands in seconds.

**Returns** `TODO`

**Return type** *UsbTransport*

**classmethod** **find\_all\_adb\_devices** (*default\_transport\_timeout\_s=None*)

Find all ADB devices attached via USB.

**Parameters** **default\_transport\_timeout\_s** (*TODO*, *None*) – Default timeout of commands in seconds.

**Returns** A generator which yields each ADB device attached via USB.

**Return type** generator

**property** **port\_path**

`TODO`

**Returns** `TODO`

**Return type** `TODO`

**property** **serial\_number**

`TODO`

**Returns** `TODO`

**Return type** `TODO`

**property** **usb\_info**

`TODO`

**Returns** `TODO`

**Return type** `TODO`

`adb_shell.transport.usb_transport.get_interface` (*setting*)

Get the class, subclass, and protocol for the given USB setting.

**Parameters** **setting** (*TODO*) – `TODO`

**Returns**

- *TODO* – `TODO`
- *TODO* – `TODO`
- *TODO* – `TODO`

`adb_shell.transport.usb_transport.interface_matcher` (*clazz, subclass, protocol*)

Returns a matcher that returns the setting with the given interface.

**Parameters**

- **clazz** (*TODO*) – `TODO`
- **subclass** (*TODO*) – `TODO`
- **protocol** (*TODO*) – `TODO`

**Returns** **matcher** – `TODO`

**Return type** function

## Module contents

### 1.1.2 Submodules

#### **adb\_shell.adb\_device module**

Implement the *AdbDevice* class, which can connect to a device and run ADB shell commands.

#### Contents

- *AdbDevice*
  - *AdbDevice.\_close()*
  - *AdbDevice.\_filesync\_flush()*
  - *AdbDevice.\_filesync\_read()*
  - *AdbDevice.\_filesync\_read\_buffered()*
  - *AdbDevice.\_filesync\_read\_until()*
  - *AdbDevice.\_filesync\_send()*
  - *AdbDevice.\_transport\_progress()*
  - *AdbDevice.\_okay()*
  - *AdbDevice.\_open()*
  - *AdbDevice.\_pull()*
  - *AdbDevice.\_push()*
  - *AdbDevice.\_read()*
  - *AdbDevice.\_read\_until()*
  - *AdbDevice.\_read\_until\_close()*
  - *AdbDevice.\_send()*
  - *AdbDevice.\_service()*
  - *AdbDevice.\_streaming\_command()*
  - *AdbDevice.\_streaming\_service()*
  - *AdbDevice.\_write()*
  - *AdbDevice.available*
  - *AdbDevice.close()*
  - *AdbDevice.connect()*
  - *AdbDevice.list()*
  - *AdbDevice.max\_chunk\_size*
  - *AdbDevice.pull()*
  - *AdbDevice.push()*
  - *AdbDevice.root()*

- `AdbDevice.shell()`
- `AdbDevice.stat()`
- `AdbDevice.streaming_shell()`

- `AdbDeviceTcp`
- `AdbDeviceUsb`

**class** adb\_shell.adb\_device.**AdbDevice** (*transport*, *default\_transport\_timeout\_s=None*, *banner=None*)

Bases: object

A class with methods for connecting to a device and executing ADB commands.

#### Parameters

- **transport** (`BaseTransport`) – A user-provided transport for communicating with the device; must be an instance of a subclass of `BaseTransport`
- **default\_transport\_timeout\_s** (`float`, `None`) – Default timeout in seconds for transport packets, or `None`
- **banner** (`str`, `bytes`, `None`) – The hostname of the machine where the Python interpreter is currently running; if it is not provided, it will be determined via `socket.gethostname()`

**Raises** `adb_shell.exceptions.InvalidTransportError` – The passed `transport` is not an instance of a subclass of `BaseTransport`

#### `_available`

Whether an ADB connection to the device has been established

**Type** bool

#### `_banner`

The hostname of the machine where the Python interpreter is currently running

**Type** bytearray, bytes

#### `_default_transport_timeout_s`

Default timeout in seconds for transport packets, or `None`

**Type** float, None

#### `_maxdata`

Maximum amount of data in an ADB packet

**Type** int

#### `_transport`

The transport that is used to connect to the device; must be a subclass of `BaseTransport`

**Type** `BaseTransport`

#### `_close` (*adb\_info*)

Send a `b'CLSE'` message.

**Warning:** This is not to be confused with the `AdbDevice.close()` method!

**Parameters** `adb_info` (`_AdbTransactionInfo`) – Info and settings for this ADB transaction

**`_filesync_flush`** (*adb\_info, filesync\_info*)

Write the data in the buffer up to `filesync_info.send_idx`, then set `filesync_info.send_idx` to 0.

**Parameters**

- **`adb_info`** (`_AdbTransactionInfo`) – Info and settings for this ADB transaction
- **`filesync_info`** (`_FileSyncTransactionInfo`) – Data and storage for this FileSync transaction

**`_filesync_read`** (*expected\_ids, adb\_info, filesync\_info, read\_data=True*)

Read ADB messages and return FileSync packets.

**Parameters**

- **`expected_ids`** (*tuple[bytes]*) – If the received header ID is not in `expected_ids`, an exception will be raised
- **`adb_info`** (`_AdbTransactionInfo`) – Info and settings for this ADB transaction
- **`filesync_info`** (`_FileSyncTransactionInfo`) – Data and storage for this FileSync transaction
- **`read_data`** (*bool*) – Whether to read the received data

**Returns**

- **`command_id`** (*bytes*) – The received header ID
- *tuple* – The contents of the header
- **`data`** (*bytearray, None*) – The received data, or `None` if `read_data` is `False`

**Raises**

- **`adb_shell.exceptions.AdbCommandFailureException`** – Command failed
- **`adb_shell.exceptions.InvalidResponseError`** – Received response was not in `expected_ids`

**`_filesync_read_buffered`** (*size, adb\_info, filesync\_info*)

Read `size` bytes of data from `self.recv_buffer`.

**Parameters**

- **`size`** (*int*) – The amount of data to read
- **`adb_info`** (`_AdbTransactionInfo`) – Info and settings for this ADB transaction
- **`filesync_info`** (`_FileSyncTransactionInfo`) – Data and storage for this FileSync transaction

**Returns** `result` – The read data

**Return type** `bytearray`

**`_filesync_read_until`** (*expected\_ids, finish\_ids, adb\_info, filesync\_info*)

Useful wrapper around `AdbDevice._filesync_read()`.

**Parameters**

- **`expected_ids`** (*tuple[bytes]*) – If the received header ID is not in `expected_ids`, an exception will be raised
- **`finish_ids`** (*tuple[bytes]*) – We will read until we find a header ID that is in `finish_ids`

- **adb\_info** (`_AdbTransactionInfo`) – Info and settings for this ADB transaction
- **filesync\_info** (`_FileSyncTransactionInfo`) – Data and storage for this FileSync transaction

#### Yields

- **cmd\_id** (*bytes*) – The received header ID
- **header** (*tuple*) – TODO
- **data** (*bytearray*) – The received data

**\_filesync\_send** (*command\_id, adb\_info, filesync\_info, data=b”, size=None*)  
Send/buffer FileSync packets.

Packets are buffered and only flushed when this connection is read from. All messages have a response from the device, so this will always get flushed.

#### Parameters

- **command\_id** (*bytes*) – Command to send.
- **adb\_info** (`_AdbTransactionInfo`) – Info and settings for this ADB transaction
- **filesync\_info** (`_FileSyncTransactionInfo`) – Data and storage for this FileSync transaction
- **data** (*str, bytes*) – Optional data to send, must set data or size.
- **size** (*int, None*) – Optionally override size from len(data).

**\_get\_transport\_timeout\_s** (*transport\_timeout\_s*)  
Use the provided `transport_timeout_s` if it is not `None`; otherwise, use `self._default_transport_timeout_s`

**Parameters** `transport_timeout_s` (*float, None*) – The potential transport timeout

**Returns** `transport_timeout_s` if it is not `None`; otherwise, `self._default_transport_timeout_s`

**Return type** float

**\_okay** (*adb\_info*)  
Send an `b'OKAY'` message.

**Parameters** `adb_info` (`_AdbTransactionInfo`) – Info and settings for this ADB transaction

**\_open** (*destination, adb\_info*)  
Opens a new connection to the device via an `b'OPEN'` message.

1. `_send()` an `b'OPEN'` command to the device that specifies the `local_id`
2. `_read()` a response from the device that includes a command, another local ID (`their_local_id`), and `remote_id`
  - If `local_id` and `their_local_id` do not match, raise an exception.
  - If the received command is `b'CLSE'`, `_read()` another response from the device
  - If the received command is not `b'OKAY'`, raise an exception
  - Set the `adb_info.local_id` and `adb_info.remote_id` attributes

#### Parameters

- **destination** (*bytes*) – `b'SERVICE:COMMAND'`
- **adb\_info** (`_AdbTransactionInfo`) – Info and settings for this ADB transaction

Raises `adb_shell.exceptions.InvalidResponseError` – Wrong `local_id` sent to us.

**\_pull** (*device\_path, stream, progress\_callback, adb\_info, filesync\_info*)

Pull a file from the device into the file-like `local_path`.

#### Parameters

- **device\_path** (*str*) – The file on the device that will be pulled
- **stream** (`_io.BytesIO`) – File-like object for writing to
- **progress\_callback** (*function, None*) – Callback method that accepts `device_path`, `bytes_written`, and `total_bytes`
- **adb\_info** (`_AdbTransactionInfo`) – Info and settings for this ADB transaction
- **filesync\_info** (`_FileSyncTransactionInfo`) – Data and storage for this FileSync transaction

**\_push** (*stream, device\_path, st\_mode, mtime, progress\_callback, adb\_info, filesync\_info*)

Push a file-like object to the device.

#### Parameters

- **stream** (`_io.BytesIO`) – File-like object for reading from
- **device\_path** (*str*) – Destination on the device to write to
- **st\_mode** (*int*) – Stat mode for the file
- **mtime** (*int*) – Modification time
- **progress\_callback** (*function, None*) – Callback method that accepts `device_path`, `bytes_written`, and `total_bytes`
- **adb\_info** (`_AdbTransactionInfo`) – Info and settings for this ADB transaction

Raises `PushFailedError` – Raised on push failure.

**\_read** (*expected\_cmds, adb\_info*)

Receive a response from the device.

1. Read a message from the device and unpack the `cmd`, `arg0`, `arg1`, `data_length`, and `data_checksum` fields
2. If `cmd` is not a recognized command in `adb_shell.constants.WIRE_TO_ID`, raise an exception
3. If the time has exceeded `read_timeout_s`, raise an exception
4. Read `data_length` bytes from the device
5. If the checksum of the read data does not match `data_checksum`, raise an exception
6. Return `command`, `arg0`, `arg1`, and `bytes(data)`

#### Parameters

- **expected\_cmds** (*list[bytes]*) – We will read packets until we encounter one whose “command” field is in `expected_cmds`
- **adb\_info** (`_AdbTransactionInfo`) – Info and settings for this ADB transaction



**Returns**

- **command** (*bytes*) – The received command, which is in `adb_shell.constants.WIRE_TO_ID` and must be in `expected_cmds`
- **arg0** (*int*) – TODO
- **arg1** (*int*) – TODO
- *bytes* – The data that was read

**Raises**

- `adb_shell.exceptions.InvalidCommandError` – Unknown command *or* never got one of the expected responses.
- `adb_shell.exceptions.InvalidChecksumError` – Received checksum does not match the expected checksum.

`_read_until` (*expected\_cmds*, *adb\_info*)

Read a packet, acknowledging any write packets.

1. Read data via `AdbDevice._read()`
2. If a b'WRTE' packet is received, send an b'OKAY' packet via `AdbDevice._okay()`
3. Return the cmd and data that were read by `AdbDevice._read()`

**Parameters**

- **expected\_cmds** (*list[bytes]*) – `AdbDevice._read()` with look for a packet whose command is in `expected_cmds`
- **adb\_info** (`_AdbTransactionInfo`) – Info and settings for this ADB transaction

**Returns**

- **cmd** (*bytes*) – The command that was received by `AdbDevice._read()`, which is in `adb_shell.constants.WIRE_TO_ID` and must be in `expected_cmds`
- **data** (*bytes*) – The data that was received by `AdbDevice._read()`

**Raises**

- `adb_shell.exceptions.InterleavedDataError` – We don't support multiple streams...
- `adb_shell.exceptions.InvalidResponseError` – Incorrect remote id.
- `adb_shell.exceptions.InvalidCommandError` – Never got one of the expected responses.

`_read_until_close` (*adb\_info*)

Yield packets until a b'CLSE' packet is received.

1. Read the cmd and data fields from a b'CLSE' or b'WRTE' packet via `AdbDevice._read_until()`
2. If cmd is b'CLSE', then send a b'CLSE' message and stop
3. Yield data and repeat

**Parameters** **adb\_info** (`_AdbTransactionInfo`) – Info and settings for this ADB transaction

**Yields data** (*bytes*) – The data that was read by `AdbDevice._read_until()`

**`_send`** (*msg, adb\_info*)

Send a message to the device.

1. Send the message header (`adb_shell.adb_message.AdbMessage.pack`)
2. Send the message data

#### Parameters

- **`msg`** (*AdbMessage*) – The data that will be sent
- **`adb_info`** (*\_AdbTransactionInfo*) – Info and settings for this ADB transaction

**`_service`** (*service, command, transport\_timeout\_s=None, read\_timeout\_s=10.0, timeout\_s=None, decode=True*)

Send an ADB command to the device.

#### Parameters

- **`service`** (*bytes*) – The ADB service to talk to (e.g., `b'shell'`)
- **`command`** (*bytes*) – The command that will be sent
- **`transport_timeout_s`** (*float, None*) – Timeout in seconds for sending and receiving packets, or `None`; see `BaseTransport.bulk_read()` and `BaseTransport.bulk_write()`
- **`read_timeout_s`** (*float*) – The total time in seconds to wait for a `b'CLSE'` or `b'OKAY'` command in `AdbDevice._read()`
- **`timeout_s`** (*float, None*) – The total time in seconds to wait for the ADB command to finish
- **`decode`** (*bool*) – Whether to decode the output to utf8 before returning

**Returns** The output of the ADB command as a string if `decode` is `True`, otherwise as bytes.

**Return type** `bytes, str`

**`_streaming_command`** (*service, command, adb\_info*)

One complete set of USB packets for a single command.

1. `_open()` a new connection to the device, where the destination parameter is `service:command`
2. Read the response data via `AdbDevice._read_until_close()`

---

**Note:** All the data is held in memory, and thus large responses will be slow and can fill up memory.

---

#### Parameters

- **`service`** (*bytes*) – The ADB service (e.g., `b'shell'`, as used by `AdbDevice.shell()`)
- **`command`** (*bytes*) – The service command
- **`adb_info`** (*\_AdbTransactionInfo*) – Info and settings for this ADB transaction

**Yields** *bytes* – The responses from the service.

**`_streaming_service`** (*service, command, transport\_timeout\_s=None, read\_timeout\_s=10.0, decode=True*)

Send an ADB command to the device, yielding each line of output.

**Parameters**

- **`service`** (*bytes*) – The ADB service to talk to (e.g., `b' shell '`)
- **`command`** (*bytes*) – The command that will be sent
- **`transport_timeout_s`** (*float, None*) – Timeout in seconds for sending and receiving packets, or `None`; see `BaseTransport.bulk_read()` and `BaseTransport.bulk_write()`
- **`read_timeout_s`** (*float*) – The total time in seconds to wait for a `b'CLSE'` or `b'OKAY'` command in `AdbDevice._read()`
- **`decode`** (*bool*) – Whether to decode the output to utf8 before returning

**Yields** *bytes, str* – The line-by-line output of the ADB command as a string if `decode` is `True`, otherwise as bytes.

**`static _transport_progress`** (*progress\_callback*)

Calls the callback with the current progress and total bytes written/received.

**Parameters** **`progress_callback`** (*function*) – Callback method that accepts `device_path`, `bytes_written`, and `total_bytes`

**`_write`** (*data, adb\_info*)

Write a packet and expect an Ack.

**Parameters**

- **`data`** (*bytes*) – The data that will be sent
- **`adb_info`** (`_AdbTransactionInfo`) – Info and settings for this ADB transaction

**property available**

Whether or not an ADB connection to the device has been established.

**Returns** `self._available`

**Return type** `bool`

**`close()`**

Close the connection via the provided transport's `close()` method.

**`connect`** (*rsa\_keys=None, transport\_timeout\_s=None, auth\_timeout\_s=10.0, read\_timeout\_s=10.0, auth\_callback=None*)

Establish an ADB connection to the device.

1. Use the transport to establish a connection
2. Send a `b'CNXN'` message
3. Unpack the `cmd`, `arg0`, `arg1`, and `banner` fields from the response
4. If `cmd` is not `b'AUTH'`, then authentication is not necessary and so we are done
5. If no `rsa_keys` are provided, raise an exception
6. Loop through our keys, signing the last `banner` that we received
  1. If the last `arg0` was not `adb_shell.constants.AUTH_TOKEN`, raise an exception
  2. Sign the last `banner` and send it in a `b'AUTH'` message

3. Unpack the `cmd`, `arg0`, and `banner` fields from the response via `adb_shell.adb_message.unpack()`
4. If `cmd` is `b'CNXN'`, set transfer maxdata and return `True`
7. None of the keys worked, so send `rsa_keys[0]`'s public key; if the response does not time out, we must have connected successfully

#### Parameters

- **rsa\_keys** (*list, None*) – A list of signers of type *CryptographySigner*, *PycryptodomeAuthSigner*, or *PythonRSASigner*
- **transport\_timeout\_s** (*float, None*) – Timeout in seconds for sending and receiving packets, or `None`; see *BaseTransport.bulk\_read()* and *BaseTransport.bulk\_write()*
- **auth\_timeout\_s** (*float, None*) – The time in seconds to wait for a `b'CNXN'` authentication response
- **read\_timeout\_s** (*float*) – The total time in seconds to wait for expected commands in *AdbDevice.\_read()*
- **auth\_callback** (*function, None*) – Function callback invoked when the connection needs to be accepted on the device

**Returns** Whether the connection was established (*AdbDevice.available*)

**Return type** `bool`

#### Raises

- *adb\_shell.exceptions.DeviceAuthError* – Device authentication required, no keys available
- *adb\_shell.exceptions.InvalidResponseError* – Invalid auth response from the device

**exec\_out** (*command, transport\_timeout\_s=None, read\_timeout\_s=10.0, timeout\_s=None, decode=True*)

Send an ADB `exec-out` command to the device.

<https://www.linux-magazine.com/Issues/2017/195/Ask-Klaus>

#### Parameters

- **command** (*str*) – The `exec-out` command that will be sent
- **transport\_timeout\_s** (*float, None*) – Timeout in seconds for sending and receiving packets, or `None`; see *BaseTransport.bulk\_read()* and *BaseTransport.bulk\_write()*
- **read\_timeout\_s** (*float*) – The total time in seconds to wait for a `b'CLSE'` or `b'OKAY'` command in *AdbDevice.\_read()*
- **timeout\_s** (*float, None*) – The total time in seconds to wait for the ADB command to finish
- **decode** (*bool*) – Whether to decode the output to utf8 before returning

**Returns** The output of the ADB `exec-out` command as a string if `decode` is `True`, otherwise as bytes.

**Return type** `bytes, str`

**list** (*device\_path*, *transport\_timeout\_s=None*, *read\_timeout\_s=10.0*)

Return a directory listing of the given path.

#### Parameters

- **device\_path** (*str*) – Directory to list.
- **transport\_timeout\_s** (*float*, *None*) – Expected timeout for any part of the pull.
- **read\_timeout\_s** (*float*) – The total time in seconds to wait for a b'CLSE' or b'OKAY' command in `AdbDevice._read()`

**Returns** **files** – Filename, mode, size, and mtime info for the files in the directory

**Return type** list[*DeviceFile*]

**property max\_chunk\_size**

Maximum chunk size for filesync operations

**Returns** Minimum value based on `adb_shell.constants.MAX_CHUNK_SIZE` and `_max_data / 2`, fallback to legacy `adb_shell.constants.MAX_PUSH_DATA`

**Return type** int

**pull** (*device\_path*, *local\_path*, *progress\_callback=None*, *transport\_timeout\_s=None*, *read\_timeout\_s=10.0*)

Pull a file from the device.

#### Parameters

- **device\_path** (*str*) – The file on the device that will be pulled
- **local\_path** (*str*) – The path to where the file will be downloaded
- **progress\_callback** (*function*, *None*) – Callback method that accepts `device_path`, `bytes_written`, and `total_bytes`
- **transport\_timeout\_s** (*float*, *None*) – Expected timeout for any part of the pull.
- **read\_timeout\_s** (*float*) – The total time in seconds to wait for a b'CLSE' or b'OKAY' command in `AdbDevice._read()`

**push** (*local\_path*, *device\_path*, *st\_mode=33272*, *mtime=0*, *progress\_callback=None*, *transport\_timeout\_s=None*, *read\_timeout\_s=10.0*)

Push a file or directory to the device.

#### Parameters

- **local\_path** (*str*) – Either a filename or a directory to push to the device.
- **device\_path** (*str*) – Destination on the device to write to.
- **st\_mode** (*int*) – Stat mode for `local_path`
- **mtime** (*int*) – Modification time to set on the file.
- **progress\_callback** (*function*, *None*) – Callback method that accepts `device_path`, `bytes_written`, and `total_bytes`
- **transport\_timeout\_s** (*float*, *None*) – Expected timeout for any part of the push.
- **read\_timeout\_s** (*float*) – The total time in seconds to wait for a b'CLSE' or b'OKAY' command in `AdbDevice._read()`

**root** (*transport\_timeout\_s=None, read\_timeout\_s=10.0, timeout\_s=None*)  
Gain root access.

The device must be rooted in order for this to work.

#### Parameters

- **transport\_timeout\_s** (*float, None*) – Timeout in seconds for sending and receiving packets, or None; see *BaseTransport.bulk\_read()* and *BaseTransport.bulk\_write()*
- **read\_timeout\_s** (*float*) – The total time in seconds to wait for a b'CLSE' or b'OKAY' command in *AdbDevice.\_read()*
- **timeout\_s** (*float, None*) – The total time in seconds to wait for the ADB command to finish

**shell** (*command, transport\_timeout\_s=None, read\_timeout\_s=10.0, timeout\_s=None, decode=True*)  
Send an ADB shell command to the device.

#### Parameters

- **command** (*str*) – The shell command that will be sent
- **transport\_timeout\_s** (*float, None*) – Timeout in seconds for sending and receiving packets, or None; see *BaseTransport.bulk\_read()* and *BaseTransport.bulk\_write()*
- **read\_timeout\_s** (*float*) – The total time in seconds to wait for a b'CLSE' or b'OKAY' command in *AdbDevice.\_read()*
- **timeout\_s** (*float, None*) – The total time in seconds to wait for the ADB command to finish
- **decode** (*bool*) – Whether to decode the output to utf8 before returning

**Returns** The output of the ADB shell command as a string if *decode* is True, otherwise as bytes.

**Return type** bytes, str

**stat** (*device\_path, transport\_timeout\_s=None, read\_timeout\_s=10.0*)  
Get a file's *stat()* information.

#### Parameters

- **device\_path** (*str*) – The file on the device for which we will get information.
- **transport\_timeout\_s** (*float, None*) – Expected timeout for any part of the pull.
- **read\_timeout\_s** (*float*) – The total time in seconds to wait for a b'CLSE' or b'OKAY' command in *AdbDevice.\_read()*

#### Returns

- **mode** (*int*) – The octal permissions for the file
- **size** (*int*) – The size of the file
- **mtime** (*int*) – The last modified time for the file

**streaming\_shell** (*command, transport\_timeout\_s=None, read\_timeout\_s=10.0, decode=True*)  
Send an ADB shell command to the device, yielding each line of output.

#### Parameters

- **command** (*str*) – The shell command that will be sent
- **transport\_timeout\_s** (*float, None*) – Timeout in seconds for sending and receiving packets, or None; see `BaseTransport.bulk_read()` and `BaseTransport.bulk_write()`
- **read\_timeout\_s** (*float*) – The total time in seconds to wait for a b'CLSE' or b'OKAY' command in `AdbDevice._read()`
- **decode** (*bool*) – Whether to decode the output to utf8 before returning

**Yields** *bytes, str* – The line-by-line output of the ADB shell command as a string if `decode` is True, otherwise as bytes.

```
class adb_shell.adb_device.AdbDeviceTcp(host, port=5555, de-
                                     fault_transport_timeout_s=None, banner=None)
Bases: adb_shell.adb_device.AdbDevice
```

A class with methods for connecting to a device via TCP and executing ADB commands.

#### Parameters

- **host** (*str*) – The address of the device; may be an IP address or a host name
- **port** (*int*) – The device port to which we are connecting (default is 5555)
- **default\_transport\_timeout\_s** (*float, None*) – Default timeout in seconds for TCP packets, or None
- **banner** (*str, bytes, None*) – The hostname of the machine where the Python interpreter is currently running; if it is not provided, it will be determined via `socket.gethostname()`

#### **`__available`**

Whether an ADB connection to the device has been established

**Type** bool

#### **`__banner`**

The hostname of the machine where the Python interpreter is currently running

**Type** bytearray, bytes

#### **`__default_transport_timeout_s`**

Default timeout in seconds for TCP packets, or None

**Type** float, None

#### **`__maxdata`**

Maximum amount of data in an ADB packet

**Type** int

#### **`__transport`**

The transport that is used to connect to the device

**Type** *TcpTransport*

```
class adb_shell.adb_device.AdbDeviceUsb(serial=None, port_path=None, de-
                                       fault_transport_timeout_s=None, banner=None)
```

Bases: *adb\_shell.adb\_device.AdbDevice*

A class with methods for connecting to a device via USB and executing ADB commands.

#### Parameters

- **serial** (*str, None*) – The USB device serial ID

- **port\_path** (*TODO, None*) – TODO
- **default\_transport\_timeout\_s** (*float, None*) – Default timeout in seconds for USB packets, or None
- **banner** (*str, bytes, None*) – The hostname of the machine where the Python interpreter is currently running; if it is not provided, it will be determined via `socket.gethostname()`

**Raises** `adb_shell.exceptions.InvalidTransportError` – Raised if package was not installed with the “usb” extras option (`pip install adb-shell[usb]`)

**`__available`**

Whether an ADB connection to the device has been established

**Type** bool

**`__banner`**

The hostname of the machine where the Python interpreter is currently running

**Type** bytearray, bytes

**`__default_transport_timeout_s`**

Default timeout in seconds for USB packets, or None

**Type** float, None

**`__maxdata`**

Maximum amount of data in an ADB packet

**Type** int

**`__transport`**

The transport that is used to connect to the device

**Type** *UsbTransport*

## adb\_shell.adb\_device\_async module

Implement the `AdbDeviceAsync` class, which can connect to a device and run ADB shell commands.

- `AdbDeviceAsync`
  - `AdbDeviceAsync._close()`
  - `AdbDeviceAsync._filesync_flush()`
  - `AdbDeviceAsync._filesync_read()`
  - `AdbDeviceAsync._filesync_read_buffered()`
  - `AdbDeviceAsync._filesync_read_until()`
  - `AdbDeviceAsync._filesync_send()`
  - `AdbDeviceAsync._transport_progress()`
  - `AdbDeviceAsync._okay()`
  - `AdbDeviceAsync._open()`
  - `AdbDeviceAsync._pull()`
  - `AdbDeviceAsync._push()`
  - `AdbDeviceAsync._read()`



- `AdbDeviceAsync._read_until()`
- `AdbDeviceAsync._read_until_close()`
- `AdbDeviceAsync._send()`
- `AdbDeviceAsync._service()`
- `AdbDeviceAsync._streaming_command()`
- `AdbDeviceAsync._streaming_service()`
- `AdbDeviceAsync._write()`
- `AdbDeviceAsync.available`
- `AdbDeviceAsync.close()`
- `AdbDeviceAsync.connect()`
- `AdbDeviceAsync.list()`
- `AdbDeviceAsync.max_chunk_size`
- `AdbDeviceAsync.pull()`
- `AdbDeviceAsync.push()`
- `AdbDeviceAsync.root()`
- `AdbDeviceAsync.shell()`
- `AdbDeviceAsync.stat()`
- `AdbDeviceAsync.streaming_shell()`

- `AdbDeviceTcpAsync`

**class** `adb_shell.adb_device_async.AdbDeviceAsync` (*transport*, *default\_transport\_timeout\_s=None*, *banner=None*)

Bases: `object`

A class with methods for connecting to a device and executing ADB commands.

#### Parameters

- **transport** (`BaseTransportAsync`) – A user-provided transport for communicating with the device; must be an instance of a subclass of `BaseTransportAsync`
- **default\_transport\_timeout\_s** (`float`, `None`) – Default timeout in seconds for transport packets, or `None`
- **banner** (`str`, `bytes`, `None`) – The hostname of the machine where the Python interpreter is currently running; if it is not provided, it will be determined via `socket.gethostname()`

**Raises** `adb_shell.exceptions.InvalidTransportError` – The passed `transport` is not an instance of a subclass of `BaseTransportAsync`

#### `_available`

Whether an ADB connection to the device has been established

**Type** `bool`

#### `_banner`

The hostname of the machine where the Python interpreter is currently running

**Type** `bytearray`, `bytes`

**`_default_transport_timeout_s`**

Default timeout in seconds for transport packets, or None

**Type** float, None

**`_maxdata`**

Maximum amount of data in an ADB packet

**Type** int

**`_transport`**

The transport that is used to connect to the device; must be a subclass of *BaseTransportAsync*

**Type** *BaseTransportAsync*

**`async _close (adb_info)`**

Send a b'CLSE' message.

**Warning:** This is not to be confused with the *AdbDeviceAsync.close()* method!

**Parameters** `adb_info` (*\_AdbTransactionInfo*) – Info and settings for this ADB transaction

**`async _filesync_flush (adb_info, filesync_info)`**

Write the data in the buffer up to `filesync_info.send_idx`, then set `filesync_info.send_idx` to 0.

**Parameters**

- `adb_info` (*\_AdbTransactionInfo*) – Info and settings for this ADB transaction
- `filesync_info` (*\_FileSyncTransactionInfo*) – Data and storage for this FileSync transaction

**`async _filesync_read (expected_ids, adb_info, filesync_info, read_data=True)`**

Read ADB messages and return FileSync packets.

**Parameters**

- `expected_ids` (*tuple[bytes]*) – If the received header ID is not in `expected_ids`, an exception will be raised
- `adb_info` (*\_AdbTransactionInfo*) – Info and settings for this ADB transaction
- `filesync_info` (*\_FileSyncTransactionInfo*) – Data and storage for this FileSync transaction
- `read_data` (*bool*) – Whether to read the received data

**Returns**

- `command_id` (*bytes*) – The received header ID
- *tuple* – The contents of the header
- `data` (*bytearray, None*) – The received data, or None if `read_data` is False

**Raises**

- *adb\_shell.exceptions.AdbCommandFailureException* – Command failed
- *adb\_shell.exceptions.InvalidResponseError* – Received response was not in `expected_ids`

**async \_filesync\_read\_buffered** (*size, adb\_info, filesync\_info*)

Read *size* bytes of data from `self.recv_buffer`.

**Parameters**

- **size** (*int*) – The amount of data to read
- **adb\_info** (`_AdbTransactionInfo`) – Info and settings for this ADB transaction
- **filesync\_info** (`_FileSyncTransactionInfo`) – Data and storage for this FileSync transaction

**Returns result** – The read data

**Return type** bytearray

**\_filesync\_read\_until** (*expected\_ids, finish\_ids, adb\_info, filesync\_info*)

Useful wrapper around `AdbDeviceAsync._filesync_read()`.

**Parameters**

- **expected\_ids** (*tuple[bytes]*) – If the received header ID is not in `expected_ids`, an exception will be raised
- **finish\_ids** (*tuple[bytes]*) – We will read until we find a header ID that is in `finish_ids`
- **adb\_info** (`_AdbTransactionInfo`) – Info and settings for this ADB transaction
- **filesync\_info** (`_FileSyncTransactionInfo`) – Data and storage for this FileSync transaction

**Yields**

- **cmd\_id** (*bytes*) – The received header ID
- **header** (*tuple*) – TODO
- **data** (*bytearray*) – The received data

**async \_filesync\_send** (*command\_id, adb\_info, filesync\_info, data=b'', size=None*)

Send/buffer FileSync packets.

Packets are buffered and only flushed when this connection is read from. All messages have a response from the device, so this will always get flushed.

**Parameters**

- **command\_id** (*bytes*) – Command to send.
- **adb\_info** (`_AdbTransactionInfo`) – Info and settings for this ADB transaction
- **filesync\_info** (`_FileSyncTransactionInfo`) – Data and storage for this FileSync transaction
- **data** (*str, bytes*) – Optional data to send, must set data or size.
- **size** (*int, None*) – Optionally override size from `len(data)`.

**\_get\_transport\_timeout\_s** (*transport\_timeout\_s*)

Use the provided `transport_timeout_s` if it is not `None`; otherwise, use `self._default_transport_timeout_s`

**Parameters transport\_timeout\_s** (*float, None*) – The potential transport timeout

**Returns** `transport_timeout_s` if it is not `None`; otherwise, `self._default_transport_timeout_s`

**Return type** float

**async \_okay** (*adb\_info*)  
Send an b'OKAY' message.

**Parameters** **adb\_info** (`_AdbTransactionInfo`) – Info and settings for this ADB transaction

**async \_open** (*destination, adb\_info*)  
Opens a new connection to the device via an b'OPEN' message.

1. `_send()` an b'OPEN' command to the device that specifies the `local_id`
2. `_read()` a response from the device that includes a command, another local ID (`their_local_id`), and `remote_id`
  - If `local_id` and `their_local_id` do not match, raise an exception.
  - If the received command is b'CLSE', `_read()` another response from the device
  - If the received command is not b'OKAY', raise an exception
  - Set the `adb_info.local_id` and `adb_info.remote_id` attributes

**Parameters**

- **destination** (*bytes*) – b'SERVICE:COMMAND'
- **adb\_info** (`_AdbTransactionInfo`) – Info and settings for this ADB transaction

**Raises** `adb_shell.exceptions.InvalidResponseError` – Wrong `local_id` sent to us.

**async \_pull** (*device\_path, stream, progress\_callback, adb\_info, filesync\_info*)  
Pull a file from the device into the file-like `local_path`.

**Parameters**

- **device\_path** (*str*) – The file on the device that will be pulled
- **stream** (`AsyncBufferedIOBase`) – File-like object for writing to
- **progress\_callback** (*function, None*) – Callback method that accepts `device_path`, `bytes_written`, and `total_bytes`
- **adb\_info** (`_AdbTransactionInfo`) – Info and settings for this ADB transaction
- **filesync\_info** (`_FileSyncTransactionInfo`) – Data and storage for this FileSync transaction

**async \_push** (*stream, device\_path, st\_mode, mtime, progress\_callback, adb\_info, filesync\_info*)  
Push a file-like object to the device.

**Parameters**

- **stream** (`AsyncBufferedReader`) – File-like object for reading from
- **device\_path** (*str*) – Destination on the device to write to
- **st\_mode** (*int*) – Stat mode for the file
- **mtime** (*int*) – Modification time
- **progress\_callback** (*function, None*) – Callback method that accepts `device_path`, `bytes_written`, and `total_bytes`
- **adb\_info** (`_AdbTransactionInfo`) – Info and settings for this ADB transaction

Raises *PushFailedError* – Raised on push failure.

**async \_read** (*expected\_cmds*, *adb\_info*)

Receive a response from the device.

1. Read a message from the device and unpack the *cmd*, *arg0*, *arg1*, *data\_length*, and *data\_checksum* fields
2. If *cmd* is not a recognized command in *adb\_shell.constants.WIRE\_TO\_ID*, raise an exception
3. If the time has exceeded *read\_timeout\_s*, raise an exception
4. Read *data\_length* bytes from the device
5. If the checksum of the read data does not match *data\_checksum*, raise an exception
6. Return *command*, *arg0*, *arg1*, and *bytes(data)*

#### Parameters

- **expected\_cmds** (*list[bytes]*) – We will read packets until we encounter one whose “command” field is in *expected\_cmds*
- **adb\_info** (*\_AdbTransactionInfo*) – Info and settings for this ADB transaction

#### Returns

- **command** (*bytes*) – The received command, which is in *adb\_shell.constants.WIRE\_TO\_ID* and must be in *expected\_cmds*
- **arg0** (*int*) – TODO
- **arg1** (*int*) – TODO
- *bytes* – The data that was read

#### Raises

- *adb\_shell.exceptions.InvalidCommandError* – Unknown command or never got one of the expected responses.
- *adb\_shell.exceptions.InvalidChecksumError* – Received checksum does not match the expected checksum.

**async \_read\_until** (*expected\_cmds*, *adb\_info*)

Read a packet, acknowledging any write packets.

1. Read data via *AdbDeviceAsync.\_read()*
2. If a b'WRTE' packet is received, send an b'OKAY' packet via *AdbDeviceAsync.\_okay()*
3. Return the *cmd* and *data* that were read by *AdbDeviceAsync.\_read()*

#### Parameters

- **expected\_cmds** (*list[bytes]*) – *AdbDeviceAsync.\_read()* with look for a packet whose command is in *expected\_cmds*
- **adb\_info** (*\_AdbTransactionInfo*) – Info and settings for this ADB transaction

#### Returns

- **cmd** (*bytes*) – The command that was received by *AdbDeviceAsync.\_read()*, which is in *adb\_shell.constants.WIRE\_TO\_ID* and must be in *expected\_cmds*

- **data** (*bytes*) – The data that was received by `AdbDeviceAsync._read()`

**Raises**

- `adb_shell.exceptions.InterleavedDataError` – We don't support multiple streams...
- `adb_shell.exceptions.InvalidResponseError` – Incorrect remote id.
- `adb_shell.exceptions.InvalidCommandError` – Never got one of the expected responses.

**`_read_until_close`** (*adb\_info*)

Yield packets until a b'CLSE' packet is received.

1. Read the `cmd` and `data` fields from a b'CLSE' or b'WRTE' packet via `AdbDeviceAsync._read_until()`
2. If `cmd` is b'CLSE', then send a b'CLSE' message and stop
3. Yield data and repeat

**Parameters** `adb_info` (`_AdbTransactionInfo`) – Info and settings for this ADB transaction

**Yields** `data` (*bytes*) – The data that was read by `AdbDeviceAsync._read_until()`

**`async _send`** (*msg, adb\_info*)

Send a message to the device.

1. Send the message header (`adb_shell.adb_message.AdbMessage.pack`)
2. Send the message data

**Parameters**

- `msg` (`AdbMessage`) – The data that will be sent
- `adb_info` (`_AdbTransactionInfo`) – Info and settings for this ADB transaction

**`async _service`** (*service, command, transport\_timeout\_s=None, read\_timeout\_s=10.0, timeout\_s=None, decode=True*)

Send an ADB command to the device.

**Parameters**

- `service` (*bytes*) – The ADB service to talk to (e.g., b'shell')
- `command` (*bytes*) – The command that will be sent
- `transport_timeout_s` (*float, None*) – Timeout in seconds for sending and receiving packets, or None; see `BaseTransportAsync.bulk_read()` and `BaseTransportAsync.bulk_write()`
- `read_timeout_s` (*float*) – The total time in seconds to wait for a b'CLSE' or b'OKAY' command in `AdbDeviceAsync._read()`
- `timeout_s` (*float, None*) – The total time in seconds to wait for the ADB command to finish
- `decode` (*bool*) – Whether to decode the output to utf8 before returning

**Returns** The output of the ADB command as a string if `decode` is True, otherwise as bytes.

**Return type** bytes, str

**`_streaming_command`** (*service, command, adb\_info*)

One complete set of USB packets for a single command.

1. `_open()` a new connection to the device, where the destination parameter is `service:command`
2. Read the response data via `AdbDeviceAsync._read_until_close()`

---

**Note:** All the data is held in memory, and thus large responses will be slow and can fill up memory.

---

#### Parameters

- **service** (*bytes*) – The ADB service (e.g., `b'shell'`, as used by `AdbDeviceAsync.shell()`)
- **command** (*bytes*) – The service command
- **adb\_info** (`_AdbTransactionInfo`) – Info and settings for this ADB transaction

**Yields** *bytes* – The responses from the service.

**`_streaming_service`** (*service, command, transport\_timeout\_s=None, read\_timeout\_s=10.0, decode=True*)

Send an ADB command to the device, yielding each line of output.

#### Parameters

- **service** (*bytes*) – The ADB service to talk to (e.g., `b'shell'`)
- **command** (*bytes*) – The command that will be sent
- **transport\_timeout\_s** (*float, None*) – Timeout in seconds for sending and receiving packets, or None; see `BaseTransportAsync.bulk_read()` and `BaseTransportAsync.bulk_write()`
- **read\_timeout\_s** (*float*) – The total time in seconds to wait for a `b'CLSE'` or `b'OKAY'` command in `AdbDeviceAsync._read()`
- **decode** (*bool*) – Whether to decode the output to utf8 before returning

**Yields** *bytes, str* – The line-by-line output of the ADB command as a string if `decode` is `True`, otherwise as bytes.

**static `_transport_progress`** (*progress\_callback*)

Calls the callback with the current progress and total bytes written/received.

**Parameters** **progress\_callback** (*function*) – Callback method that accepts `device_path`, `bytes_written`, and `total_bytes`

**async `_write`** (*data, adb\_info*)

Write a packet and expect an Ack.

#### Parameters

- **data** (*bytes*) – The data that will be sent
- **adb\_info** (`_AdbTransactionInfo`) – Info and settings for this ADB transaction

**property available**

Whether or not an ADB connection to the device has been established.

**Returns** `self._available`

**Return type** `bool`

**async close()**

Close the connection via the provided transport's `close()` method.

**async connect** (*rsa\_keys=None, transport\_timeout\_s=None, auth\_timeout\_s=10.0, read\_timeout\_s=10.0, auth\_callback=None*)

Establish an ADB connection to the device.

1. Use the transport to establish a connection
2. Send a b'CNXN' message
3. Unpack the `cmd`, `arg0`, `arg1`, and `banner` fields from the response
4. If `cmd` is not b'AUTH', then authentication is not necessary and so we are done
5. If no `rsa_keys` are provided, raise an exception
6. Loop through our keys, signing the last `banner` that we received
  1. If the last `arg0` was not `adb_shell.constants.AUTH_TOKEN`, raise an exception
  2. Sign the last `banner` and send it in an b'AUTH' message
  3. Unpack the `cmd`, `arg0`, and `banner` fields from the response via `adb_shell.adb_message.unpack()`
  4. If `cmd` is b'CNXN', set transfer `maxdata` size and return `True`
7. None of the keys worked, so send `rsa_keys[0]`'s public key; if the response does not time out, we must have connected successfully

**Parameters**

- **rsa\_keys** (*list, None*) – A list of signers of type *CryptographySigner*, *PycryptodomeAuthSigner*, or *PythonRSASigner*
- **transport\_timeout\_s** (*float, None*) – Timeout in seconds for sending and receiving packets, or `None`; see *BaseTransportAsync.bulk\_read()* and *BaseTransportAsync.bulk\_write()*
- **auth\_timeout\_s** (*float, None*) – The time in seconds to wait for a b'CNXN' authentication response
- **read\_timeout\_s** (*float*) – The total time in seconds to wait for expected commands in *AdbDeviceAsync.\_read()*
- **auth\_callback** (*function, None*) – Function callback invoked when the connection needs to be accepted on the device

**Returns** Whether the connection was established (*AdbDeviceAsync.available*)

**Return type** `bool`

**Raises**

- *adb\_shell.exceptions.DeviceAuthError* – Device authentication required, no keys available
- *adb\_shell.exceptions.InvalidResponseError* – Invalid auth response from the device

**async exec\_out** (*command, transport\_timeout\_s=None, read\_timeout\_s=10.0, timeout\_s=None, decode=True*)

Send an ADB `exec-out` command to the device.

<https://www.linux-magazine.com/Issues/2017/195/Ask-Klaus>



**Parameters**

- **command** (*str*) – The exec-out command that will be sent
- **transport\_timeout\_s** (*float, None*) – Timeout in seconds for sending and receiving packets, or None; see *BaseTransport.bulk\_read()* and *BaseTransport.bulk\_write()*
- **read\_timeout\_s** (*float*) – The total time in seconds to wait for a b'CLSE' or b'OKAY' command in *AdbDevice.\_read()*
- **timeout\_s** (*float, None*) – The total time in seconds to wait for the ADB command to finish
- **decode** (*bool*) – Whether to decode the output to utf8 before returning

**Returns** The output of the ADB exec-out command as a string if *decode* is True, otherwise as bytes.

**Return type** bytes, str

**async list** (*device\_path, transport\_timeout\_s=None, read\_timeout\_s=10.0*)

Return a directory listing of the given path.

**Parameters**

- **device\_path** (*str*) – Directory to list.
- **transport\_timeout\_s** (*float, None*) – Expected timeout for any part of the pull.
- **read\_timeout\_s** (*float*) – The total time in seconds to wait for a b'CLSE' or b'OKAY' command in *AdbDeviceAsync.\_read()*

**Returns** files – Filename, mode, size, and mtime info for the files in the directory

**Return type** list[*DeviceFile*]

**property max\_chunk\_size**

Maximum chunk size for filesync operations

**Returns** Minimum value based on *adb\_shell.constants.MAX\_CHUNK\_SIZE* and *\_max\_data / 2*, fallback to legacy *adb\_shell.constants.MAX\_PUSH\_DATA*

**Return type** int

**async pull** (*device\_path, local\_path, progress\_callback=None, transport\_timeout\_s=None, read\_timeout\_s=10.0*)

Pull a file from the device.

**Parameters**

- **device\_path** (*str*) – The file on the device that will be pulled
- **local\_path** (*str*) – The path to where the file will be downloaded
- **progress\_callback** (*function, None*) – Callback method that accepts *device\_path*, *bytes\_written*, and *total\_bytes*
- **transport\_timeout\_s** (*float, None*) – Expected timeout for any part of the pull.
- **read\_timeout\_s** (*float*) – The total time in seconds to wait for a b'CLSE' or b'OKAY' command in *AdbDevice.\_read()*

**async push** (*local\_path*, *device\_path*, *st\_mode*=33272, *mtime*=0, *progress\_callback*=None, *transport\_timeout\_s*=None, *read\_timeout\_s*=10.0)

Push a file or directory to the device.

#### Parameters

- **local\_path** (*str*) – Either a filename or a directory to push to the device
- **device\_path** (*str*) – Destination on the device to write to
- **st\_mode** (*int*) – Stat mode for *local\_path*
- **mtime** (*int*) – Modification time to set on the file
- **progress\_callback** (*function*, None) – Callback method that accepts *device\_path*, *bytes\_written*, and *total\_bytes*
- **transport\_timeout\_s** (*float*, None) – Expected timeout for any part of the push
- **read\_timeout\_s** (*float*) – The total time in seconds to wait for a b'CLSE' or b'OKAY' command in `AdbDeviceAsync._read()`

**async root** (*transport\_timeout\_s*=None, *read\_timeout\_s*=10.0, *timeout\_s*=None)

Gain root access.

The device must be rooted in order for this to work.

#### Parameters

- **transport\_timeout\_s** (*float*, None) – Timeout in seconds for sending and receiving packets, or None; see `BaseTransportAsync.bulk_read()` and `BaseTransportAsync.bulk_write()`
- **read\_timeout\_s** (*float*) – The total time in seconds to wait for a b'CLSE' or b'OKAY' command in `AdbDeviceAsync._read()`
- **timeout\_s** (*float*, None) – The total time in seconds to wait for the ADB command to finish

**async shell** (*command*, *transport\_timeout\_s*=None, *read\_timeout\_s*=10.0, *timeout\_s*=None, *decode*=True)

Send an ADB shell command to the device.

#### Parameters

- **command** (*str*) – The shell command that will be sent
- **transport\_timeout\_s** (*float*, None) – Timeout in seconds for sending and receiving packets, or None; see `BaseTransportAsync.bulk_read()` and `BaseTransportAsync.bulk_write()`
- **read\_timeout\_s** (*float*) – The total time in seconds to wait for a b'CLSE' or b'OKAY' command in `AdbDeviceAsync._read()`
- **timeout\_s** (*float*, None) – The total time in seconds to wait for the ADB command to finish
- **decode** (*bool*) – Whether to decode the output to utf8 before returning

**Returns** The output of the ADB shell command as a string if *decode* is True, otherwise as bytes.

**Return type** bytes, str

**async\_stat** (*device\_path*, *transport\_timeout\_s=None*, *read\_timeout\_s=10.0*)

Get a file's `stat()` information.

#### Parameters

- **device\_path** (*str*) – The file on the device for which we will get information.
- **transport\_timeout\_s** (*float*, *None*) – Expected timeout for any part of the pull.
- **read\_timeout\_s** (*float*) – The total time in seconds to wait for a `b'CLSE'` or `b'OKAY'` command in `AdbDeviceAsync._read()`

#### Returns

- **mode** (*int*) – The octal permissions for the file
- **size** (*int*) – The size of the file
- **mtime** (*int*) – The last modified time for the file

**streaming\_shell** (*command*, *transport\_timeout\_s=None*, *read\_timeout\_s=10.0*, *decode=True*)

Send an ADB shell command to the device, yielding each line of output.

#### Parameters

- **command** (*str*) – The shell command that will be sent
- **transport\_timeout\_s** (*float*, *None*) – Timeout in seconds for sending and receiving packets, or `None`; see `BaseTransportAsync.bulk_read()` and `BaseTransportAsync.bulk_write()`
- **read\_timeout\_s** (*float*) – The total time in seconds to wait for a `b'CLSE'` or `b'OKAY'` command in `AdbDeviceAsync._read()`
- **decode** (*bool*) – Whether to decode the output to utf8 before returning

**Yields** *bytes*, *str* – The line-by-line output of the ADB shell command as a string if `decode` is `True`, otherwise as bytes.

```
class adb_shell.adb_device_async.AdbDeviceTcpAsync(host, port=5555, default_transport_timeout_s=None,
                                                banner=None)
```

Bases: `adb_shell.adb_device_async.AdbDeviceAsync`

A class with methods for connecting to a device via TCP and executing ADB commands.

#### Parameters

- **host** (*str*) – The address of the device; may be an IP address or a host name
- **port** (*int*) – The device port to which we are connecting (default is 5555)
- **default\_transport\_timeout\_s** (*float*, *None*) – Default timeout in seconds for TCP packets, or `None`
- **banner** (*str*, *bytes*, *None*) – The hostname of the machine where the Python interpreter is currently running; if it is not provided, it will be determined via `socket.gethostname()`

#### **`_available`**

Whether an ADB connection to the device has been established

**Type** `bool`

#### **`_banner`**

The hostname of the machine where the Python interpreter is currently running

**Type** bytearray, bytes

**`_default_transport_timeout_s`**

Default timeout in seconds for TCP packets, or None

**Type** float, None

**`_maxdata`**

Maximum amount of data in an ADB packet

**Type** int

**`_transport`**

The transport that is used to connect to the device

**Type** *TcpTransportAsync*

## adb\_shell.adb\_message module

Functions and an *AdbMessage* class for packing and unpacking ADB messages.

### Contents

- *AdbMessage*
  - *AdbMessage.checksum*
  - *AdbMessage.pack()*
- *checksum()*
- *int\_to\_cmd()*
- *unpack()*

**class** adb\_shell.adb\_message.**AdbMessage** (*command, arg0, arg1, data=b''*)

Bases: object

A helper class for packing ADB messages.

#### Parameters

- **command** (*bytes*) – A command; examples used in this package include `adb_shell.constants.AUTH`, `adb_shell.constants.CNXXN`, `adb_shell.constants.CLSE`, `adb_shell.constants.OPEN`, and `adb_shell.constants.OKAY`
- **arg0** (*int*) – Usually the local ID, but *connect()* and *connect()* provide `adb_shell.constants.VERSION`, `adb_shell.constants.AUTH_SIGNATURE`, and `adb_shell.constants.AUTH_RSAPUBLICKEY`
- **arg1** (*int*) – Usually the remote ID, but *connect()* and *connect()* provide `adb_shell.constants.MAX_ADB_DATA`
- **data** (*bytes*) – The data that will be sent

#### arg0

Usually the local ID, but *connect()* and *connect()* provide `adb_shell.constants.VERSION`, `adb_shell.constants.AUTH_SIGNATURE`, and `adb_shell.constants.AUTH_RSAPUBLICKEY`

**Type** int

**arg1**

Usually the remote ID, but `connect()` and `connect()` provide `adb_shell.constants.MAX_ADB_DATA`

**Type** int

**command**

The input parameter `command` converted to an integer via `adb_shell.constants.ID_TO_WIRE`

**Type** int

**data**

The data that will be sent

**Type** bytes

**magic**

`self.command` with its bits flipped; in other words, `self.command + self.magic == 2**32 - 1`

**Type** int

**property checksum**

Return `checksum(self.data)`

**Returns** The checksum of `self.data`

**Return type** int

**pack()**

Returns this message in an over-the-wire format.

**Returns** The message packed into the format required by ADB

**Return type** bytes

`adb_shell.adb_message.checksum(data)`

Calculate the checksum of the provided data.

**Parameters** `data` (*bytearray*, *bytes*, *str*) – The data

**Returns** The checksum

**Return type** int

`adb_shell.adb_message.int_to_cmd(n)`

Convert from an integer (4 bytes) to an ADB command.

**Parameters** `n` (*int*) – The integer that will be converted to an ADB command

**Returns** The ADB command (e.g., 'CNXN')

**Return type** str

`adb_shell.adb_message.unpack(message)`

Unpack a received ADB message.

**Parameters** `message` (*bytes*) – The received message

**Returns**

- `cmd` (*int*) – The ADB command
- `arg0` (*int*) – TODO
- `arg1` (*int*) – TODO

- **data\_length** (*int*) – The length of the data sent by the device (used by `adb_shell.AdbDevice._read()` and `adb_shell.AdbDeviceAsync._read()`)
- **data\_checksum** (*int*) – The checksum of the data sent by the device

**Raises** `ValueError` – Unable to unpack the ADB command.

## adb\_shell.constants module

Constants used throughout the code.

`adb_shell.constants.AUTH_RSAPUBLICKEY = 3`  
AUTH constant for `arg0`

`adb_shell.constants.AUTH_SIGNATURE = 2`  
AUTH constant for `arg0`

`adb_shell.constants.AUTH_TOKEN = 1`  
AUTH constant for `arg0`

`adb_shell.constants.CLASS = 255`  
From `adb.h`

`adb_shell.constants.DEFAULT_AUTH_TIMEOUT_S = 10.0`  
Default authentication timeout (in s) for `adb_shell.AdbDevice.connect()` and `adb_shell.AdbDeviceAsync.connect()`

`adb_shell.constants.DEFAULT_PUSH_MODE = 33272`  
Default mode for pushed files.

`adb_shell.constants.DEFAULT_READ_TIMEOUT_S = 10.0`  
Default total timeout (in s) for `adb_shell.AdbDevice._read()`, `adb_shell.AdbDevice._read_until()`, `adb_shell.AdbDeviceAsync._read()`, and `adb_shell.AdbDeviceAsync._read_until()`

`adb_shell.constants.FILESYNC_IDS = (b'DATA', b'DENT', b'DONE', b'FAIL', b'LIST', b'OKAY', b'DONTKNOW')`  
Commands that are recognized by `adb_shell.AdbDevice._filesync_read()` and `adb_shell.AdbDeviceAsync._filesync_read()`

`adb_shell.constants.FILESYNC_ID_TO_WIRE = {b'DATA': 1096040772, b'DENT': 1414415684, b'DONE': 1145980243, b'FAIL': 1145980243, b'LIST': 1145980243, b'OKAY': 1145980243, b'DONTKNOW': 1145980243}`  
A dictionary where the keys are the commands in `FILESYNC_IDS` and the values are the keys converted to integers

`adb_shell.constants.FILESYNC_LIST_FORMAT = b'<5I'`  
The format for FileSync “list” messages

`adb_shell.constants.FILESYNC_PULL_FORMAT = b'<2I'`  
The format for FileSync “pull” messages

`adb_shell.constants.FILESYNC_PUSH_FORMAT = b'<2I'`  
The format for FileSync “push” messages

`adb_shell.constants.FILESYNC_STAT_FORMAT = b'<4I'`  
The format for FileSync “stat” messages

`adb_shell.constants.FILESYNC_WIRE_TO_ID = {1096040772: b'DATA', 1145980243: b'SEND', 1145980243: b'DONE', 1145980243: b'FAIL', 1145980243: b'LIST', 1145980243: b'OKAY', 1145980243: b'DONTKNOW'}`  
A dictionary where the keys are integers and the values are their corresponding commands (type = bytes) from `FILESYNC_IDS`

`adb_shell.constants.IDS = (b'AUTH', b'CLSE', b'CNXN', b'OKAY', b'OPEN', b'SYNC', b'WRTE')`  
 Commands that are recognized by `adb_shell.adb_device.AdbDevice._read()` and `adb_shell.adb_device_async.AdbDeviceAsync._read()`

`adb_shell.constants.ID_TO_WIRE = {b'AUTH': 1213486401, b'CLSE': 1163086915, b'CNXN': 13144...}`  
 A dictionary where the keys are the commands in *IDS* and the values are the keys converted to integers

`adb_shell.constants.MAX_ADB_DATA = 1048576`  
`//android.googleusercontent.com/platform/system/core+/master/adb/adb.h`  
**Type** Maximum amount of data in an ADB packet. According to  
**Type** https

`adb_shell.constants.MAX_CHUNK_SIZE = 65536`  
`//android.googleusercontent.com/platform/system/core+/master/adb/SYNC.TXT`  
**Type** Maximum chunk size. According to https

`adb_shell.constants.MAX_PUSH_DATA = 2048`  
 Maximum size of a filesync DATA packet. Default size.

`adb_shell.constants.MESSAGE_FORMAT = b'<6I'`  
 An ADB message is 6 words in little-endian.

`adb_shell.constants.MESSAGE_SIZE = 24`  
 The size of an ADB message

`adb_shell.constants.PROTOCOL = 1`  
 From adb.h

`adb_shell.constants.SUBCLASS = 66`  
 From adb.h

`adb_shell.constants.VERSION = 16777216`  
 ADB protocol version.

`adb_shell.constants.WIRE_TO_ID = {1129208147: b'SYNC', 1163086915: b'CLSE', 1163154007: ...}`  
 A dictionary where the keys are integers and the values are their corresponding commands (type = bytes) from *IDS*

## adb\_shell.exceptions module

ADB-related exceptions.

**exception** `adb_shell.exceptions.AdbCommandFailureException`

Bases: `Exception`

A b'FAIL' packet was received.

**exception** `adb_shell.exceptions.AdbConnectionError`

Bases: `Exception`

ADB command not sent because a connection to the device has not been established.

**exception** `adb_shell.exceptions.AdbTimeoutError`

Bases: `Exception`

ADB command did not complete within the specified time.

**exception** `adb_shell.exceptions.DeviceAuthError` (*message*, \*args)

Bases: `Exception`

Device authentication failed.

**exception** `adb_shell.exceptions.DevicePathInvalidError`

Bases: Exception

A file command was passed an invalid path.

**exception** `adb_shell.exceptions.InterleavedDataError`

Bases: Exception

We only support command sent serially.

**exception** `adb_shell.exceptions.InvalidChecksumError`

Bases: Exception

Checksum of data didn't match expected checksum.

**exception** `adb_shell.exceptions.InvalidCommandError`

Bases: Exception

Got an invalid command.

**exception** `adb_shell.exceptions.InvalidResponseError`

Bases: Exception

Got an invalid response to our command.

**exception** `adb_shell.exceptions.InvalidTransportError`

Bases: Exception

The provided transport does not implement the necessary methods: `close`, `connect`, `bulk_read`, and `bulk_write`.

**exception** `adb_shell.exceptions.PushFailedError`

Bases: Exception

Pushing a file failed for some reason.

**exception** `adb_shell.exceptions.TcpTimeoutException`

Bases: Exception

TCP connection timed read/write operation exceeded the allowed time.

**exception** `adb_shell.exceptions.UsbDeviceNotFoundError`

Bases: Exception

TODO

**exception** `adb_shell.exceptions.UsbReadFailedError` (*msg*, *usb\_error*)

Bases: Exception

TODO

### Parameters

- **msg** (*str*) – The error message
- **usb\_error** (*libusb1.USBError*) – An exception from `libusb1`

**usb\_error**

An exception from `libusb1`

**Type** `libusb1.USBError`

**exception** `adb_shell.exceptions.UsbWriteFailedError`

Bases: Exception

`adb_shell.transport.usb_transport.UsbTransport.bulk_write()` failed.



## adb\_shell.hidden\_helpers module

Implement helpers for the *AdbDevice* and *AdbDeviceAsync* classes.

### Contents

- *\_AdbTransactionInfo*
- *\_FileSyncTransactionInfo*
  - *\_FileSyncTransactionInfo.can\_add\_to\_send\_buffer()*
- *get\_banner()*
- *get\_files\_to\_push()*

**class** adb\_shell.hidden\_helpers.**DeviceFile** (*filename, mode, size, mtime*)

Bases: tuple

**\_asdict** ()

Return a new OrderedDict which maps field names to their values.

**\_field\_defaults** = {}

**\_fields** = ('filename', 'mode', 'size', 'mtime')

**\_fields\_defaults** = {}

**classmethod** **\_make** (*iterable*)

Make a new DeviceFile object from a sequence or iterable

**\_replace** (*\*\*kws*)

Return a new DeviceFile object replacing specified fields with new values

**property** **filename**

Alias for field number 0

**property** **mode**

Alias for field number 1

**property** **mtime**

Alias for field number 3

**property** **size**

Alias for field number 2

**class** adb\_shell.hidden\_helpers.**\_AdbTransactionInfo** (*local\_id, remote\_id, transport\_timeout\_s=None, read\_timeout\_s=10.0, timeout\_s=None*)

Bases: object

A class for storing info and settings used during a single ADB “transaction.”

Note that if `timeout_s` is not `None`, then:

```
self.transport_timeout_s <= self.read_timeout_s <= self.timeout_s
```

If `timeout_s` is `None`, the first inequality still applies.

#### Parameters

- **local\_id** (*int*) – The ID for the sender (i.e., the device running this code)

- **remote\_id** (*int*) – The ID for the recipient
- **transport\_timeout\_s** (*float, None*) – Timeout in seconds for sending and receiving packets, or None; see *BaseTransport.bulk\_read()*, *BaseTransport.bulk\_write()*, *BaseTransportAsync.bulk\_read()*, and *BaseTransportAsync.bulk\_write()*
- **read\_timeout\_s** (*float*) – The total time in seconds to wait for a command in `expected_cmds` in *AdbDevice.\_read()* and *AdbDeviceAsync.\_read()*
- **timeout\_s** (*float, None*) – The total time in seconds to wait for the ADB command to finish

**local\_id**

The ID for the sender (i.e., the device running this code)

**Type** int

**read\_timeout\_s**

The total time in seconds to wait for a command in `expected_cmds` in *AdbDevice.\_read()* and *AdbDeviceAsync.\_read()*

**Type** float

**remote\_id**

The ID for the recipient

**Type** int

**timeout\_s**

The total time in seconds to wait for the ADB command to finish

**Type** float, None

**transport\_timeout\_s**

Timeout in seconds for sending and receiving packets, or None; see *BaseTransport.bulk\_read()*, *BaseTransport.bulk\_write()*, *BaseTransportAsync.bulk\_read()*, and *BaseTransportAsync.bulk\_write()*

**Type** float, None

**class** adb\_shell.hidden\_helpers.\_FileSyncTransactionInfo (*recv\_message\_format*,  
*maxdata=1048576*)

Bases: object

A class for storing info used during a single FileSync “transaction.”

**Parameters**

- **recv\_message\_format** (*bytes*) – The FileSync message format
- **maxdata** (*int*) – Maximum amount of data in an ADB packet

**\_maxdata**

Maximum amount of data in an ADB packet

**Type** int

**recv\_buffer**

A buffer for storing received data

**Type** bytearray

**recv\_message\_format**

The FileSync message format

**Type** bytes

**recv\_message\_size**

The FileSync message size

**Type** int

**send\_buffer**

A buffer for storing data to be sent

**Type** bytearray

**send\_idx**

The index in `recv_buffer` that will be the start of the next data packet sent

**Type** int

**can\_add\_to\_send\_buffer** (*data\_len*)

Determine whether *data\_len* bytes of data can be added to the send buffer without exceeding `constants.MAX_ADB_DATA`.

**Parameters** *data\_len* (*int*) – The length of the data to be potentially added to the send buffer (not including the length of its header)

**Returns** Whether *data\_len* bytes of data can be added to the send buffer without exceeding `constants.MAX_ADB_DATA`

**Return type** bool

`adb_shell.hidden_helpers.get_banner()`

Get the banner that will be signed in `adb_shell.adb_device.AdbDevice.connect()` / `adb_shell.adb_device_async.AdbDeviceAsync.connect()`.

**Returns** The hostname, or “unknown” if it could not be determined

**Return type** bytearray

`adb_shell.hidden_helpers.get_files_to_push(local_path, device_path)`

Get a list of the file(s) to push.

**Parameters**

- **local\_path** (*str*) – A path to a local file or directory
- **device\_path** (*str*) – A path to a file or directory on the device

**Returns**

- **local\_path\_is\_dir** (*bool*) – Whether or not `local_path` is a directory
- **local\_paths** (*list[str]*) – A list of the file(s) to push
- **device\_paths** (*list[str]*) – A list of destination paths on the device that corresponds to `local_paths`

### 1.1.3 Module contents

ADB shell functionality.

This Python package implements ADB shell and FileSync functionality. It originated from `python-adb`.



## INSTALLATION

```
pip install adb-shell
```

### 2.1 Async

To utilize the async version of this code, you must install into a Python 3.7+ environment via:

```
pip install adb-shell[async]
```

### 2.2 USB Support (Experimental)

To connect to a device via USB, install this package via:

```
pip install adb-shell[usb]
```



## EXAMPLE USAGE

(Based on androidtv/adb\_manager.py)

```
from adb_shell.adb_device import AdbDeviceTcp, AdbDeviceUsb
from adb_shell.auth.sign_pythonrsa import PythonRSASigner

# Load the public and private keys
adbkey = 'path/to/adbkey'
with open(adbkey) as f:
    priv = f.read()
    with open(adbkey + '.pub') as f:
        pub = f.read()
signer = PythonRSASigner(pub, priv)

# Connect
device1 = AdbDeviceTcp('192.168.0.222', 5555, default_transport_timeout_s=9.)
device1.connect(rsa_keys=[signer], auth_timeout_s=0.1)

# Connect via USB (package must be installed via `pip install adb-shell[usb]`)
device2 = AdbDeviceUsb()
device2.connect(rsa_keys=[signer], auth_timeout_s=0.1)

# Send a shell command
response1 = device1.shell('echo TEST1')
response2 = device2.shell('echo TEST2')
```





## INDICES AND TABLES

- genindex
- modindex
- search



## PYTHON MODULE INDEX

### a

- adb\_shell, 47
- adb\_shell.adb\_device, 16
- adb\_shell.adb\_device\_async, 28
- adb\_shell.adb\_message, 40
- adb\_shell.auth, 6
  - adb\_shell.auth.keygen, 1
  - adb\_shell.auth.sign\_cryptography, 3
  - adb\_shell.auth.sign\_pycryptodome, 3
  - adb\_shell.auth.sign\_pythonsrsa, 4
- adb\_shell.constants, 42
- adb\_shell.exceptions, 43
- adb\_shell.hidden\_helpers, 45
- adb\_shell.transport, 16
  - adb\_shell.transport.base\_transport, 6
  - adb\_shell.transport.base\_transport\_async, 7
  - adb\_shell.transport.tcp\_transport, 8
  - adb\_shell.transport.tcp\_transport\_async, 9
  - adb\_shell.transport.usb\_transport, 10



## Symbols

\_Accum (class in adb\_shell.auth.sign\_pythonrsa), 5  
 \_AdbTransactionInfo (class in adb\_shell.hidden\_helpers), 45  
 \_FileSyncTransactionInfo (class in adb\_shell.hidden\_helpers), 46  
 \_HANDLE\_CACHE (adb\_shell.transport.usb\_transport.UsbTransport attribute), 12  
 \_HANDLE\_CACHE\_LOCK (adb\_shell.transport.usb\_transport.UsbTransport attribute), 12  
 \_abc\_impl (adb\_shell.transport.base\_transport.BaseTransport attribute), 6  
 \_abc\_impl (adb\_shell.transport.base\_transport\_async.BaseTransportAsync attribute), 7  
 \_abc\_impl (adb\_shell.transport.tcp\_transport.TcpTransport attribute), 8  
 \_abc\_impl (adb\_shell.transport.tcp\_transport\_async.TcpTransportAsync attribute), 10  
 \_abc\_impl (adb\_shell.transport.usb\_transport.UsbTransport attribute), 12  
 \_asdict () (adb\_shell.hidden\_helpers.DeviceFile method), 45  
 \_available (adb\_shell.adb\_device.AdbDevice attribute), 17  
 \_available (adb\_shell.adb\_device.AdbDeviceTcp attribute), 27  
 \_available (adb\_shell.adb\_device.AdbDeviceUsb attribute), 28  
 \_available (adb\_shell.adb\_device\_async.AdbDeviceAsync attribute), 29  
 \_available (adb\_shell.adb\_device\_async.AdbDeviceTcpAsync attribute), 39  
 \_banner (adb\_shell.adb\_device.AdbDevice attribute), 17  
 \_banner (adb\_shell.adb\_device.AdbDeviceTcp attribute), 27  
 \_banner (adb\_shell.adb\_device.AdbDeviceUsb attribute), 28  
 \_banner (adb\_shell.adb\_device\_async.AdbDeviceAsync attribute), 29  
 \_banner (adb\_shell.adb\_device\_async.AdbDeviceTcpAsync attribute), 39  
 \_buf (adb\_shell.auth.sign\_pythonrsa.\_Accum attribute), 5  
 \_close () (adb\_shell.adb\_device.AdbDevice method), 17  
 \_close () (adb\_shell.adb\_device\_async.AdbDeviceAsync method), 30  
 \_connection (adb\_shell.transport.tcp\_transport.TcpTransport attribute), 8  
 \_default\_transport\_timeout\_s (adb\_shell.adb\_device.AdbDevice attribute), 17  
 \_default\_transport\_timeout\_s (adb\_shell.adb\_device.AdbDeviceTcp attribute), 27  
 \_default\_transport\_timeout\_s (adb\_shell.adb\_device.AdbDeviceUsb attribute), 28  
 \_default\_transport\_timeout\_s (adb\_shell.adb\_device\_async.AdbDeviceAsync attribute), 30  
 \_default\_transport\_timeout\_s (adb\_shell.adb\_device\_async.AdbDeviceTcpAsync attribute), 40  
 \_default\_transport\_timeout\_s (adb\_shell.transport.usb\_transport.UsbTransport attribute), 11  
 \_device (adb\_shell.transport.usb\_transport.UsbTransport attribute), 11  
 field\_defaults (adb\_shell.hidden\_helpers.DeviceFile attribute), 45  
 fields (adb\_shell.hidden\_helpers.DeviceFile attribute), 45  
 \_fields\_defaults (adb\_shell.hidden\_helpers.DeviceFile attribute), 45  
 \_filesync\_flush () (adb\_shell.adb\_device.AdbDevice method), 17  
 \_filesync\_flush () (adb\_shell.adb\_device\_async.AdbDeviceAsync method), 30  
 \_filesync\_read () (adb\_shell.adb\_device.AdbDevice method), 18

\_filesync\_read() (*adb\_shell.adb\_device\_async.AdbDeviceAsync* attribute), 40  
     *method*), 30  
 \_filesync\_read\_buffered() (*adb\_shell.adb\_device.AdbDevice* *method*), 18  
 \_filesync\_read\_buffered() (*adb\_shell.adb\_device\_async.AdbDeviceAsync* *method*), 30  
 \_filesync\_read\_until() (*adb\_shell.adb\_device.AdbDevice* *method*), 18  
 \_filesync\_read\_until() (*adb\_shell.adb\_device\_async.AdbDeviceAsync* *method*), 31  
 \_filesync\_send() (*adb\_shell.adb\_device.AdbDevice* *method*), 19  
 \_filesync\_send() (*adb\_shell.adb\_device\_async.AdbDeviceAsync* attribute), 9  
     *method*), 31  
 \_find() (*adb\_shell.transport.usb\_transport.UsbTransport* *class method*), 12  
 \_find\_and\_open() (*adb\_shell.transport.usb\_transport.UsbTransport* *class method*), 12  
 \_find\_devices() (*adb\_shell.transport.usb\_transport.UsbTransport* *class method*), 12  
 \_find\_first() (*adb\_shell.transport.usb\_transport.UsbTransport* *class method*), 13  
 \_flush\_buffers() (*adb\_shell.transport.usb\_transport.UsbTransport* *method*), 13  
 \_get\_transport\_timeout\_s() (*adb\_shell.adb\_device.AdbDevice* *method*), 19  
 \_get\_transport\_timeout\_s() (*adb\_shell.adb\_device\_async.AdbDeviceAsync* *method*), 31  
 \_host (*adb\_shell.transport.tcp\_transport.TcpTransport* attribute), 8  
 \_host (*adb\_shell.transport.tcp\_transport\_async.TcpTransportAsync* attribute), 9  
 \_interface\_number (*adb\_shell.transport.usb\_transport.UsbTransport* attribute), 11  
 \_load\_rsa\_private\_key() (*in module adb\_shell.auth.sign\_pythonrsa*), 6  
 \_make() (*adb\_shell.hidden\_helpers.DeviceFile* *class method*), 45  
 \_max\_read\_packet\_len (*adb\_shell.transport.usb\_transport.UsbTransport* attribute), 12  
 \_maxdata (*adb\_shell.adb\_device.AdbDevice* attribute), 17  
 \_maxdata (*adb\_shell.adb\_device.AdbDeviceTcp* attribute), 27  
 \_maxdata (*adb\_shell.adb\_device.AdbDeviceUsb* attribute), 28  
 \_maxdata (*adb\_shell.adb\_device\_async.AdbDeviceAsync* attribute), 30  
 \_maxdata (*adb\_shell.adb\_device\_async.AdbDeviceTcpAsync* attribute), 46  
 \_okay() (*adb\_shell.adb\_device.AdbDevice* *method*), 19  
 \_okay() (*adb\_shell.adb\_device\_async.AdbDeviceAsync* *method*), 32  
 \_open() (*adb\_shell.adb\_device.AdbDevice* *method*), 19  
 \_open() (*adb\_shell.adb\_device\_async.AdbDeviceAsync* *method*), 32  
 \_open() (*adb\_shell.transport.usb\_transport.UsbTransport* *method*), 13  
 \_port (*adb\_shell.transport.tcp\_transport.TcpTransport* attribute), 8  
 \_port (*adb\_shell.transport.tcp\_transport\_async.TcpTransportAsync* attribute), 9  
 \_port\_path\_matcher() (*adb\_shell.transport.usb\_transport.UsbTransport* *class method*), 13  
 \_pull() (*adb\_shell.adb\_device\_async.AdbDeviceAsync* *method*), 32  
 \_push() (*adb\_shell.adb\_device.AdbDevice* *method*), 20  
 \_push() (*adb\_shell.adb\_device\_async.AdbDeviceAsync* *method*), 32  
 \_read() (*adb\_shell.adb\_device\_async.AdbDeviceAsync* *method*), 33  
 \_read\_endpoint (*adb\_shell.transport.usb\_transport.UsbTransport* attribute), 12  
 \_read\_until() (*adb\_shell.adb\_device.AdbDevice* *method*), 21  
 \_read\_until() (*adb\_shell.adb\_device\_async.AdbDeviceAsync* *method*), 33  
 \_read\_until\_close() (*adb\_shell.adb\_device.AdbDevice* *method*), 21  
 \_read\_until\_close() (*adb\_shell.adb\_device\_async.AdbDeviceAsync* *method*), 34  
 \_reader (*adb\_shell.transport.tcp\_transport\_async.TcpTransportAsync* attribute), 9  
 \_replace() (*adb\_shell.hidden\_helpers.DeviceFile* *method*), 45  
 \_send() (*adb\_shell.adb\_device.AdbDevice* *method*), 22  
 \_send() (*adb\_shell.adb\_device\_async.AdbDeviceAsync* *method*), 34  
 \_serial\_matcher() (*adb\_shell.transport.usb\_transport.UsbTransport* *class method*), 13  
 \_service() (*adb\_shell.adb\_device.AdbDevice* *method*), 22  
 \_service() (*adb\_shell.adb\_device\_async.AdbDeviceAsync* *method*), 34  
 \_setting (*adb\_shell.transport.usb\_transport.UsbTransport* attribute), 12

\_streaming\_command() (adb\_shell.adb\_device.AdbDevice method), 22  
 \_streaming\_command() (adb\_shell.adb\_device\_async.AdbDeviceAsync method), 35  
 \_streaming\_service() (adb\_shell.adb\_device.AdbDevice method), 22  
 \_streaming\_service() (adb\_shell.adb\_device\_async.AdbDeviceAsync method), 35  
 \_timeout\_ms() (adb\_shell.transport.usb\_transport.UsbTransport method), 14  
 \_to\_bytes() (in module adb\_shell.auth.keygen), 2  
 \_transport (adb\_shell.adb\_device.AdbDevice attribute), 17  
 \_transport (adb\_shell.adb\_device.AdbDeviceTcp attribute), 27  
 \_transport (adb\_shell.adb\_device.AdbDeviceUsb attribute), 28  
 \_transport (adb\_shell.adb\_device\_async.AdbDeviceAsync attribute), 30  
 \_transport (adb\_shell.adb\_device\_async.AdbDeviceTcpAsync attribute), 40  
 \_transport (adb\_shell.transport.usb\_transport.UsbTransport attribute), 11  
 \_transport\_progress() (adb\_shell.adb\_device.AdbDevice static method), 23  
 \_transport\_progress() (adb\_shell.adb\_device\_async.AdbDeviceAsync static method), 35  
 \_usb\_info (adb\_shell.transport.usb\_transport.UsbTransport attribute), 12  
 \_write() (adb\_shell.adb\_device.AdbDevice method), 23  
 \_write() (adb\_shell.adb\_device\_async.AdbDeviceAsync method), 35  
 \_write\_endpoint (adb\_shell.transport.usb\_transport.UsbTransport attribute), 12  
 \_writer (adb\_shell.transport.tcp\_transport\_async.TcpTransportAsync attribute), 10  
 adb\_shell.constants (module), 42  
 adb\_shell.exceptions (module), 43  
 adb\_shell.hidden\_helpers (module), 45  
 adb\_shell.transport (module), 16  
 adb\_shell.transport.base\_transport (module), 6  
 adb\_shell.transport.base\_transport\_async (module), 7  
 adb\_shell.transport.tcp\_transport (module), 8  
 adb\_shell.transport.tcp\_transport\_async (module), 9  
 adb\_shell.transport.usb\_transport (module), 10  
 AdbCommandFailureException, 43  
 AdbConnectionError, 43  
 AdbDevice (class in adb\_shell.adb\_device), 17  
 AdbDeviceAsync (class in adb\_shell.adb\_device\_async), 29  
 AdbDeviceTcp (class in adb\_shell.adb\_device), 27  
 AdbDeviceTcpAsync (class in adb\_shell.adb\_device\_async), 39  
 AdbDeviceUsb (class in adb\_shell.adb\_device), 27  
 AdbMessage (class in adb\_shell.adb\_message), 40  
 AdbTimeoutError, 43  
 ANDROID\_PUBKEY\_MODULUS\_SIZE (in module adb\_shell.auth.keygen), 2  
 ANDROID\_PUBKEY\_MODULUS\_SIZE\_WORDS (in module adb\_shell.auth.keygen), 2  
 ANDROID\_RSAPUBLICKEY\_STRUCT (in module adb\_shell.auth.keygen), 2  
 arg0 (adb\_shell.adb\_message.AdbMessage attribute), 40  
 arg1 (adb\_shell.adb\_message.AdbMessage attribute), 40  
 AUTH\_RSAPUBLICKEY (in module adb\_shell.constants), 42  
 AUTH\_SIGNATURE (in module adb\_shell.constants), 42  
 AUTH\_TOKEN (in module adb\_shell.constants), 42  
 available() (adb\_shell.adb\_device.AdbDevice property), 23  
 available() (adb\_shell.adb\_device\_async.AdbDeviceAsync property), 35

## A

adb\_shell (module), 47  
 adb\_shell.adb\_device (module), 16  
 adb\_shell.adb\_device\_async (module), 28  
 adb\_shell.adb\_message (module), 40  
 adb\_shell.auth (module), 6  
 adb\_shell.auth.keygen (module), 1  
 adb\_shell.auth.sign\_cryptography (module), 3  
 adb\_shell.auth.sign\_pycryptodome (module), 3  
 adb\_shell.auth.sign\_pythonsrsa (module), 4

## B

BaseTransport (class in adb\_shell.transport.base\_transport), 6  
 BaseTransportAsync (class in adb\_shell.transport.base\_transport\_async), 7  
 bulk\_read() (adb\_shell.transport.base\_transport.BaseTransport method), 6  
 bulk\_read() (adb\_shell.transport.base\_transport\_async.BaseTransport method), 7

bulk\_read() (*adb\_shell.transport.tcp\_transport.TcpTransport* *method*), 8  
*adb\_shell.transport.tcp\_transport\_async.TcpTransportAsync* *method*), 10  
*adb\_shell.transport.usb\_transport.UsbTransport* *method*), 14  
*adb\_shell.transport.base\_transport.BaseTransport* *method*), 7  
*adb\_shell.transport.base\_transport\_async.BaseTransportAsync* *method*), 7  
*adb\_shell.transport.tcp\_transport.TcpTransport* *method*), 8  
*adb\_shell.transport.tcp\_transport\_async.TcpTransportAsync* *method*), 10  
*adb\_shell.transport.usb\_transport.UsbTransport* *method*), 14

**C**

can\_add\_to\_send\_buffer() (*adb\_shell.hidden\_helpers.\_FileSyncTransactionInfo* *method*), 47  
checksum() (*adb\_shell.adb\_message.AdbMessage* *property*), 41  
checksum() (*in module adb\_shell.adb\_message*), 41  
CLASS (*in module adb\_shell.constants*), 42  
close() (*adb\_shell.adb\_device.AdbDevice* *method*), 23  
close() (*adb\_shell.adb\_device\_async.AdbDeviceAsync* *method*), 36  
close() (*adb\_shell.transport.base\_transport.BaseTransport* *method*), 7  
close() (*adb\_shell.transport.base\_transport\_async.BaseTransportAsync* *method*), 7  
close() (*adb\_shell.transport.tcp\_transport.TcpTransport* *method*), 9  
close() (*adb\_shell.transport.tcp\_transport\_async.TcpTransportAsync* *method*), 10  
close() (*adb\_shell.transport.usb\_transport.UsbTransport* *method*), 14  
command (*adb\_shell.adb\_message.AdbMessage* *attribute*), 41  
connect() (*adb\_shell.adb\_device.AdbDevice* *method*), 23  
connect() (*adb\_shell.adb\_device\_async.AdbDeviceAsync* *method*), 36  
connect() (*adb\_shell.transport.base\_transport.BaseTransport* *method*), 7  
connect() (*adb\_shell.transport.base\_transport\_async.BaseTransportAsync* *method*), 8  
connect() (*adb\_shell.transport.tcp\_transport.TcpTransport* *method*), 9  
connect() (*adb\_shell.transport.tcp\_transport\_async.TcpTransportAsync* *method*), 10  
connect() (*adb\_shell.transport.usb\_transport.UsbTransport* *method*), 14

CryptographySigner (class in *adb\_shell.auth.sign\_cryptography*), 3  
D  
AdbMessage (adb\_shell.adb\_message.AdbMessage attribute), 41  
decrypt\_publickey() (in module *adb\_shell.auth.keygen*), 2  
decrypt\_privatekey\_file() (in module *adb\_shell.auth.keygen*), 2  
DEFAULT\_AUTH\_TIMEOUT\_S (in module *adb\_shell.constants*), 42  
DEFAULT\_USB\_MODE (in module *adb\_shell.constants*), 42  
DEFAULT\_READ\_TIMEOUT\_S (in module *adb\_shell.constants*), 42  
DEFAULT\_TIMEOUT\_S (in module *adb\_shell.transport.usb\_transport*), 11  
DeviceAuthError, 43  
DeviceFile (class in *adb\_shell.hidden\_helpers*), 45  
DevicePathInvalidError, 43  
digest() (*adb\_shell.auth.sign\_pythonrsa.\_Accum* *method*), 5  
E  
encode\_publickey() (in module *adb\_shell.auth.keygen*), 2  
exec\_out() (*adb\_shell.adb\_device.AdbDevice* *method*), 24  
exec\_out() (*adb\_shell.adb\_device\_async.AdbDeviceAsync* *method*), 36  
F  
filename() (*adb\_shell.hidden\_helpers.DeviceFile* *property*), 45  
FILESYNC\_ID\_TO\_WIRE (in module *adb\_shell.constants*), 42  
FILESYNC\_IDS (in module *adb\_shell.constants*), 42  
FILESYNC\_LIST\_FORMAT (in module *adb\_shell.constants*), 42  
FILESYNC\_PULL\_FORMAT (in module *adb\_shell.constants*), 42  
FILESYNC\_PUSH\_FORMAT (in module *adb\_shell.constants*), 42  
FILESYNC\_STAT\_FORMAT (in module *adb\_shell.constants*), 42  
FILESYNC\_WIRE\_TO\_ID (in module *adb\_shell.constants*), 42  
find\_adb() (*adb\_shell.transport.usb\_transport.UsbTransport* *class method*), 14  
find\_all\_adb\_devices() (*adb\_shell.transport.usb\_transport.UsbTransport* *class method*), 15



- FromRSAKeyPath() (*adb\_shell.auth.sign\_pythonrsa.PythonRSAKeyPath* class method), 5
- G**
- get\_banner() (*in module adb\_shell.hidden\_helpers*), 47
- get\_files\_to\_push() (*in module adb\_shell.hidden\_helpers*), 47
- get\_interface() (*in module adb\_shell.transport.usb\_transport*), 15
- get\_user\_info() (*in module adb\_shell.auth.keygen*), 2
- GetPublicKey() (*adb\_shell.auth.sign\_cryptography.CryptographySigner* method), 3
- GetPublicKey() (*adb\_shell.auth.sign\_pycryptodome.PycryptodomeAuthSigner* method), 4
- GetPublicKey() (*adb\_shell.auth.sign\_pythonrsa.PythonRSAKeyPath* method), 5
- I**
- ID\_TO\_WIRE (*in module adb\_shell.constants*), 43
- IDS (*in module adb\_shell.constants*), 42
- int\_to\_cmd() (*in module adb\_shell.adb\_message*), 41
- interface\_matcher() (*in module adb\_shell.transport.usb\_transport*), 15
- InterleavedDataError, 44
- InvalidChecksumError, 44
- InvalidCommandError, 44
- InvalidResponseError, 44
- InvalidTransportError, 44
- K**
- keygen() (*in module adb\_shell.auth.keygen*), 2
- L**
- list() (*adb\_shell.adb\_device.AdbDevice* method), 24
- list() (*adb\_shell.adb\_device\_async.AdbDeviceAsync* method), 37
- local\_id(*adb\_shell.hidden\_helpers.\_AdbTransactionInfo* attribute), 46
- M**
- magic (*adb\_shell.adb\_message.AdbMessage* attribute), 41
- MAX\_ADB\_DATA (*in module adb\_shell.constants*), 43
- MAX\_CHUNK\_SIZE (*in module adb\_shell.constants*), 43
- max\_chunk\_size() (*adb\_shell.adb\_device.AdbDevice* property), 25
- max\_chunk\_size() (*adb\_shell.adb\_device\_async.AdbDeviceAsync* property), 37
- MAX\_PUSH\_DATA (*in module adb\_shell.constants*), 43
- MESSAGE\_FORMAT (*in module adb\_shell.constants*), 43
- MAX\_RSA\_SIGNATURE\_SIZE (*in module adb\_shell.constants*), 43
- mode() (*adb\_shell.hidden\_helpers.DeviceFile* property), 45
- mtime() (*adb\_shell.hidden\_helpers.DeviceFile* property), 45
- P**
- pack() (*adb\_shell.adb\_message.AdbMessage* method), 41
- port\_path() (*adb\_shell.transport.usb\_transport.UsbTransport* property), 15
- priv\_key(*adb\_shell.auth.sign\_pythonrsa.PythonRSAKeyPath* attribute), 5
- PROTOCOL (*in module adb\_shell.constants*), 43
- public\_key(*adb\_shell.auth.sign\_pythonrsa.PythonRSAKeyPath* attribute), 5
- public\_key(*adb\_shell.auth.sign\_cryptography.CryptographySigner* attribute), 3
- public\_key(*adb\_shell.auth.sign\_pycryptodome.PycryptodomeAuthSigner* attribute), 4
- pull() (*adb\_shell.adb\_device.AdbDevice* method), 25
- pull() (*adb\_shell.adb\_device\_async.AdbDeviceAsync* method), 37
- push() (*adb\_shell.adb\_device.AdbDevice* method), 25
- push() (*adb\_shell.adb\_device\_async.AdbDeviceAsync* method), 37
- PushFailedError, 44
- PycryptodomeAuthSigner (*class in adb\_shell.auth.sign\_pycryptodome*), 4
- PythonRSAKeyPath (*class in adb\_shell.auth.sign\_pythonrsa*), 4
- R**
- read\_timeout\_s(*adb\_shell.hidden\_helpers.\_AdbTransactionInfo* attribute), 46
- recv\_buffer(*adb\_shell.hidden\_helpers.\_FileSyncTransactionInfo* attribute), 46
- recv\_message\_format (*adb\_shell.hidden\_helpers.\_FileSyncTransactionInfo* attribute), 46
- recv\_message\_size (*adb\_shell.hidden\_helpers.\_FileSyncTransactionInfo* attribute), 47
- remote\_id(*adb\_shell.hidden\_helpers.\_AdbTransactionInfo* attribute), 46
- root() (*adb\_shell.adb\_device.AdbDevice* method), 25
- root() (*adb\_shell.adb\_device\_async.AdbDeviceAsync* method), 38
- rsa\_key(*adb\_shell.auth.sign\_cryptography.CryptographySigner* attribute), 3
- rsa\_key(*adb\_shell.auth.sign\_pycryptodome.PycryptodomeAuthSigner* attribute), 4

## S

`send_buffer` (*adb\_shell.hidden\_helpers.\_FileSyncTransactionInfo* attribute), 47  
`send_idx` (*adb\_shell.hidden\_helpers.\_FileSyncTransactionInfo* attribute), 47  
`serial_number` () (*adb\_shell.transport.usb\_transport.UsbTransport* property), 15  
`shell` () (*adb\_shell.adb\_device.AdbDevice* method), 26  
`shell` () (*adb\_shell.adb\_device\_async.AdbDeviceAsync* method), 38  
`Sign` () (*adb\_shell.auth.sign\_cryptography.CryptographySigner* method), 3  
`Sign` () (*adb\_shell.auth.sign\_pycryptodome.PycryptodomeAuthSigner* method), 4  
`Sign` () (*adb\_shell.auth.sign\_pythonrsa.PythonRSASigner* method), 5  
`size` () (*adb\_shell.hidden\_helpers.DeviceFile* property), 45  
`stat` () (*adb\_shell.adb\_device.AdbDevice* method), 26  
`stat` () (*adb\_shell.adb\_device\_async.AdbDeviceAsync* method), 38  
`streaming_shell` ()  
     (*adb\_shell.adb\_device.AdbDevice* method), 26  
`streaming_shell` ()  
     (*adb\_shell.adb\_device\_async.AdbDeviceAsync* method), 39  
SUBCLASS (in module *adb\_shell.constants*), 43

## T

*TcpTimeoutException*, 44  
*TcpTransport* (class in *adb\_shell.transport.tcp\_transport*), 8  
*TcpTransportAsync* (class in *adb\_shell.transport.tcp\_transport\_async*), 9  
`timeout_s` (*adb\_shell.hidden\_helpers.\_AdbTransactionInfo* attribute), 46  
`transport_timeout_s`  
     (*adb\_shell.hidden\_helpers.\_AdbTransactionInfo* attribute), 46

## U

`unpack` () (in module *adb\_shell.adb\_message*), 41  
`update` () (*adb\_shell.auth.sign\_pythonrsa.\_Accum* method), 6  
`usb_error` (*adb\_shell.exceptions.UsbReadFailedError* attribute), 44  
`usb_info` () (*adb\_shell.transport.usb\_transport.UsbTransport* property), 15  
*UsbDeviceNotFoundError*, 44  
*UsbReadFailedError*, 44  
*UsbTransport* (class in *adb\_shell.transport.usb\_transport*), 11  
*UsbWriteFailedError*, 44

## V

*VENDOR\_ID* (in module *adb\_shell.constants*), 43

## W

*WIRE\_TO\_ID* (in module *adb\_shell.constants*), 43

`write_public_keyfile` () (in module *adb\_shell.auth.keygen*), 3